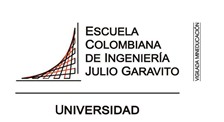
****

**Escuela Colombiana de Ingeniería Julio Garavito**

**Programación Orientada a Objetos 2024-2**

**Laboratorio VI**

**Cristian Santiago Pedraza Rodríguez**

**Andersson David Sánchez Méndez**

**22 de noviembre de 2024**

# PROGRAMACIÓN ORIENTADA A OBJETOS

**Persistencia 2024-02**

# Laboratorio 6/6 [ :) ]

## OBJETIVOS

1. Completar el código de un proyecto considerando requisitos funcionales.
2. Diseñar y construir los métodos básicos de manejo de archivos: abrir, guardar, importar y exportar.
3. Controlar las excepciones generadas al trabajar con archivos.
4. Experimentar las prácticas XP : Only one pair [integrates code at a time](http://www.extremeprogramming.org/rules/sequential.html).

Use [collective ownership](http://www.extremeprogramming.org/rules/collective.html).

# ENTREGA

* Incluyan en un archivo .zip los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.
* Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada en los espacios preparados para tal fin.
* En el espacio de entrega de avance deben indicar los logros y los problemas pendientes por resolver.

## DESARROLLO

**Preparando**

En este laboratorio vamos a extender el proyecto replicate adicionando un menú barra con las opciones básicas de entrada-salida y las opciones estándar nuevo y salir.

1. En su directorio descarguen la versión del proyecto realizado por ustedes para el laboratorio 03 y preparen el ambiente para trabajar desde **CONSOLA**
2. Ejecuten el programa, revisen la funcionalidad.

## Creando la maqueta

**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

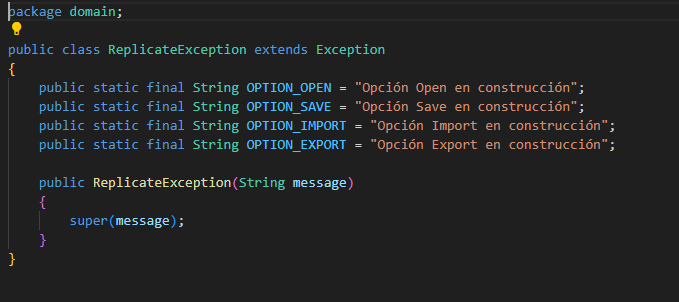
En este punto vamos a construir la maqueta correspondiente a esta extensión siguiendo el patrón MVC.

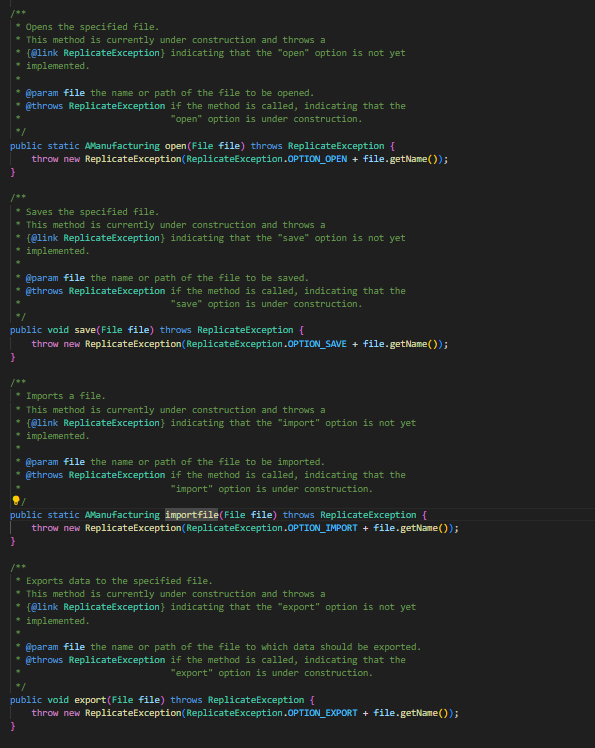
1. **MODELO:** Preparen en la clase fachada del modelo los métodos correspondientes a las cuatro opciones básicas de entrada-salida (open, save, import y export). Los métodos deben simplemente propagar una replicateException con el mensaje de “Opción nombreOpción en construcción”. Archivo nombreArchivo”. Los métodos deben tener un parámetro File.

Al crear los 4 métodos que encierran a las 4 opciones con su respectiva excepción, y estando en esa clase de excepción todos los atributos con static final para no estar mandando el mensaje en donde se lanza, se propaga o se atiende la excepción, entonces se evidencia con la captura del código la creación de esto:

Open e import son de tipo AManuFacturing, ya que, uno abre archivos para convertirlos en tipo Objeto(de tipo AManuFacturing); mientras que por el otro lado, save y export son void porque lo que hace es convertir ese objeto(menú con toda la lógica) a un archivo.

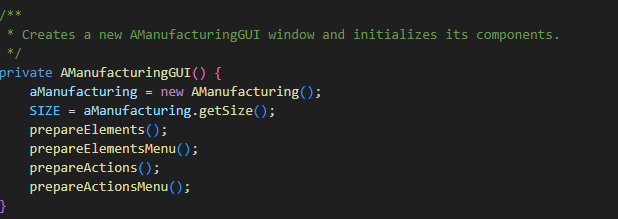
También open e import son static porque solo debe ser útil para crearse una vez dentro de la clase de dominio(AManuFacturing).





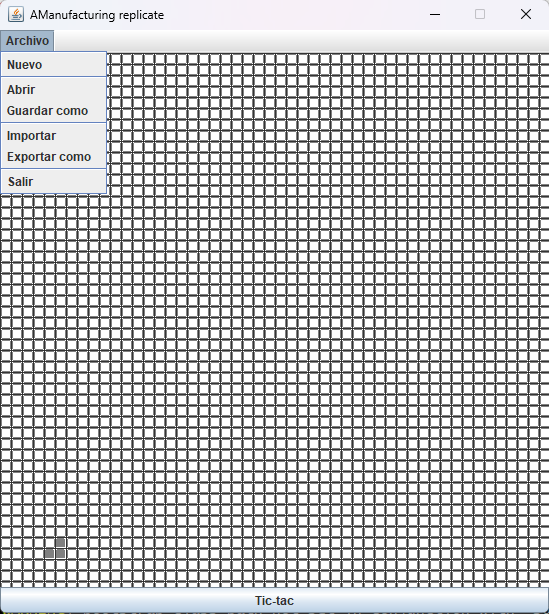
1. **VISTA :** Construyan un menú barra que ofrezca, además de las opciones básicas de entrada-salida, las opciones estándar de nuevo y salir (Nuevo, Abrir, Guardar como, Importar, Exportar como, Salir). No olviden incluir los separadores. Para esto creen el método prepareElementsMenu. Capturen la pantalla del menú.

Partiendo de la definición de prepareElementsMenu(), entonces se establecieron las opciones establecidas con los separadores:



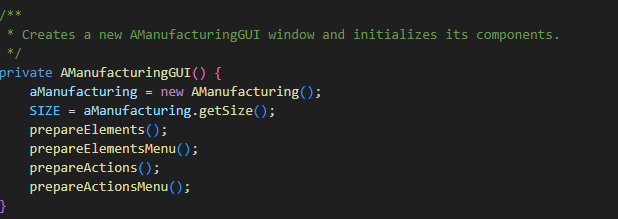


Captura del menú:



1. **CONTROLADOR:** Construyan los oyentes correspondientes a las seis opciones. Para esto creen el método prepareActionsMenu y los métodos base del controlador (optionOpen, optionSave, optionImport, optionExport, optionNew, optionExit), En las opciones que lo requieran usen un FileChooser y atiendan la excepción. Estos métodos llaman el método correspondiente de la capa de dominio que por ahora sólo lanza una excepción. Ejecuten las diferentes acciones del menú y para cada una de ellas capture una pantalla significativa.

Partiendo de lo establecido, en AManuFacturingGUI, los controladores se establecen llamando a los métodos de la fachada de dominio, atendiendo excepciones si es necesario, igual que el JFileChooser, y si atienden con try…catch



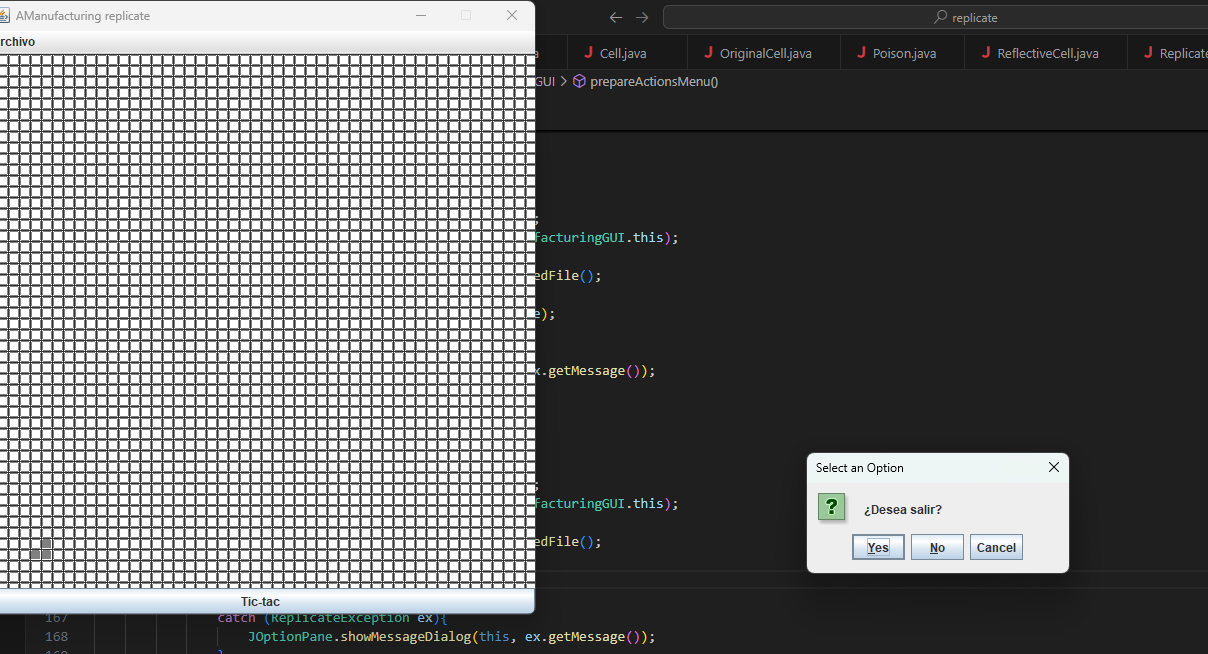




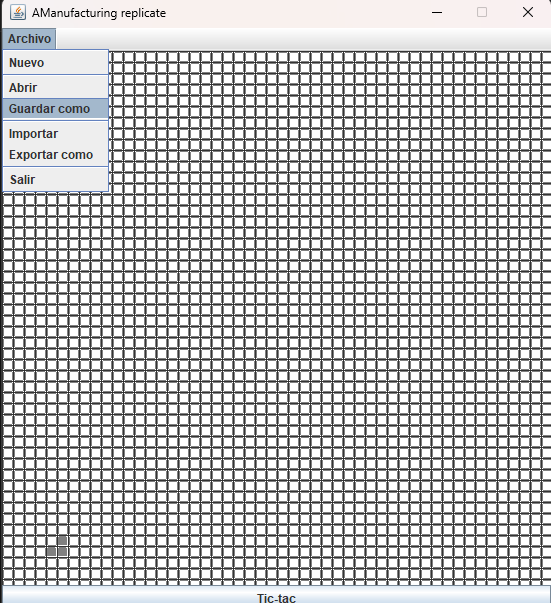
Nos damos cuenta que optionNew, optionImport, optionExport están en construcción porque en los siguientes puntos se implementan, en este solo está el de optionSave, optionOpen, optionExit.

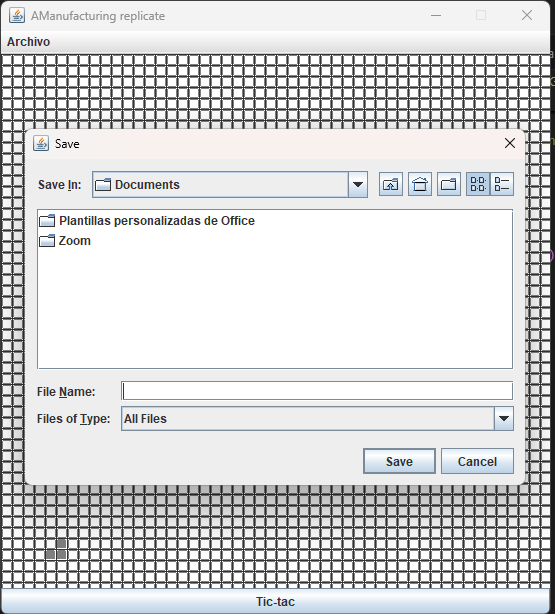
Por tanto, las capturas significativas por ahora para open, save y exit teniendo en cuenta JFileChooser son:

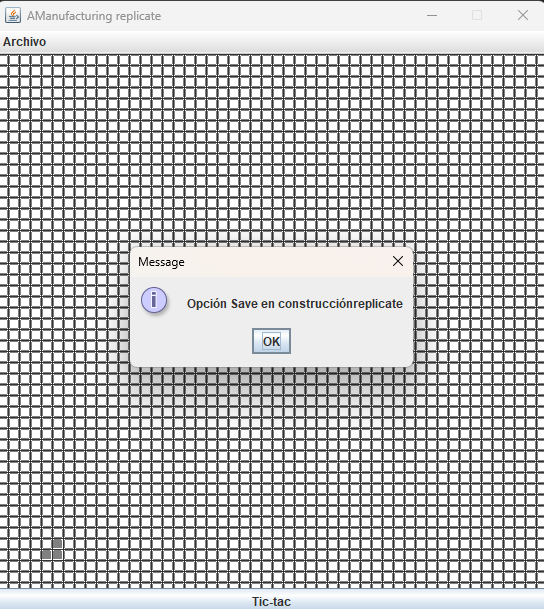
* **Para exit:**



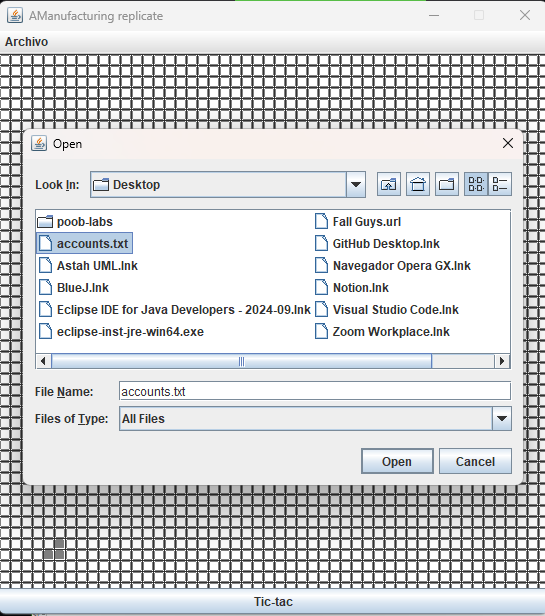
* **Para save:**

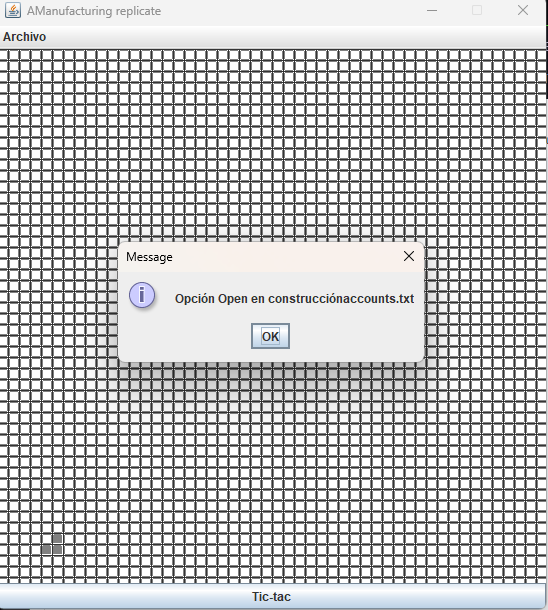
****

****

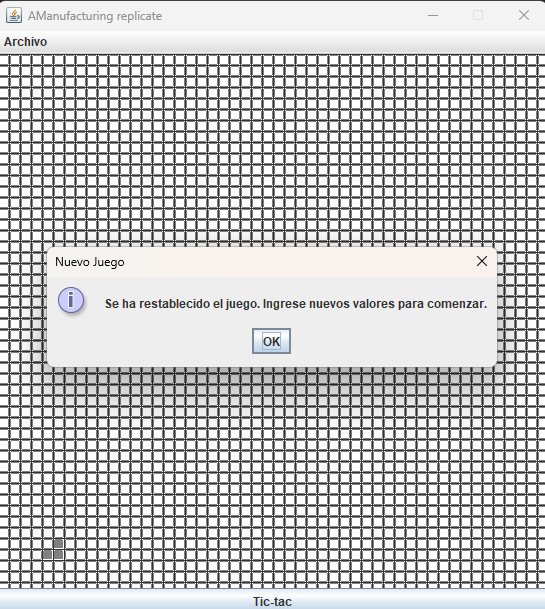
****

* **Para open:**

****

****

* **Para new: Solo está el mensaje, pero hace nada.**

****

* **Para import y export: todavía no hay funcionalidad, solo está el botón en el menú, pero hacen nada.**

****

## Implementando salir y nuevo

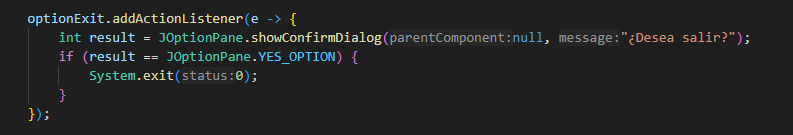
**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

Las opciones salir y nuevo van a ofrecer los dos servicios estándar de las aplicaciones. El primero no requiere ir a capa de dominio y el segundo sí.

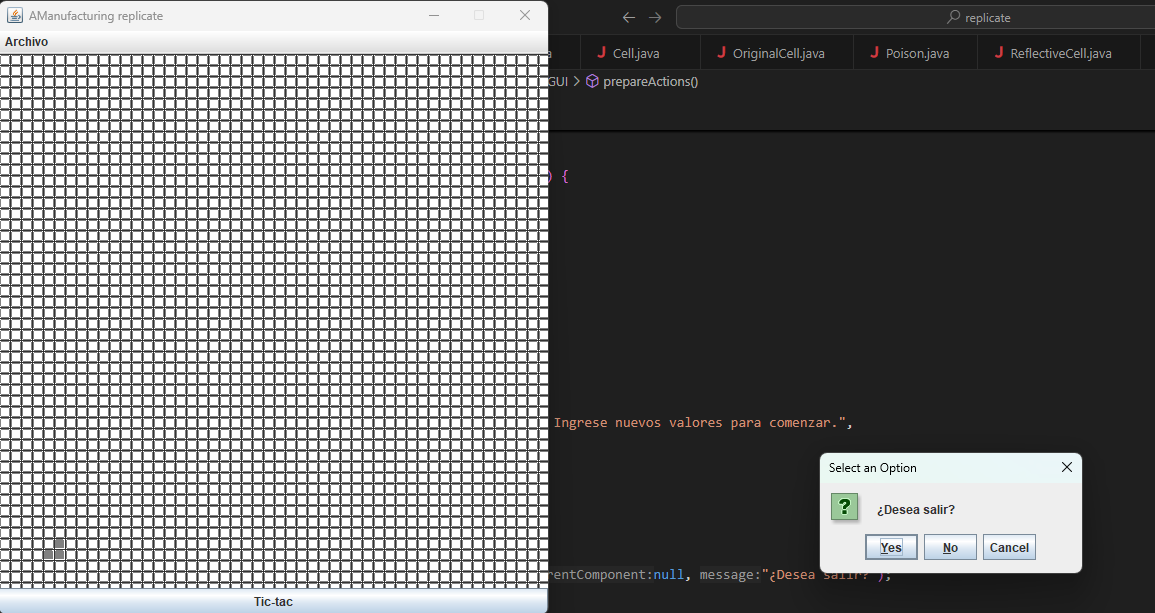
1. Construyan el método optionExit que hace que se termine la aplicación. No es necesario incluir confirmación.

Este método se creó anteriormente con confirmación, así como en el lab5; se incluyó el mensaje, ¿Quiere salir? Con opciones de Sí, No, Cancel; también esa misma confirmación se puso cuando se le da clic en X. TODO SE HACE EN PRESENTATION.



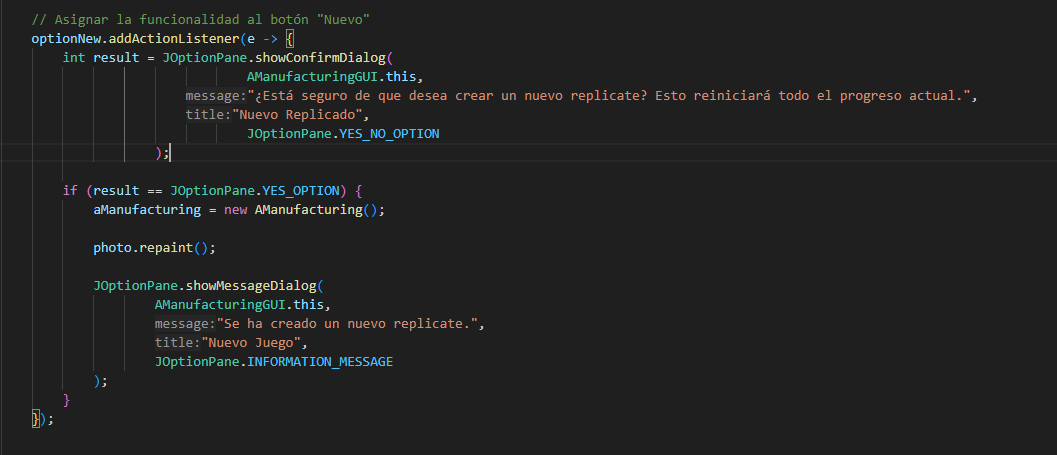


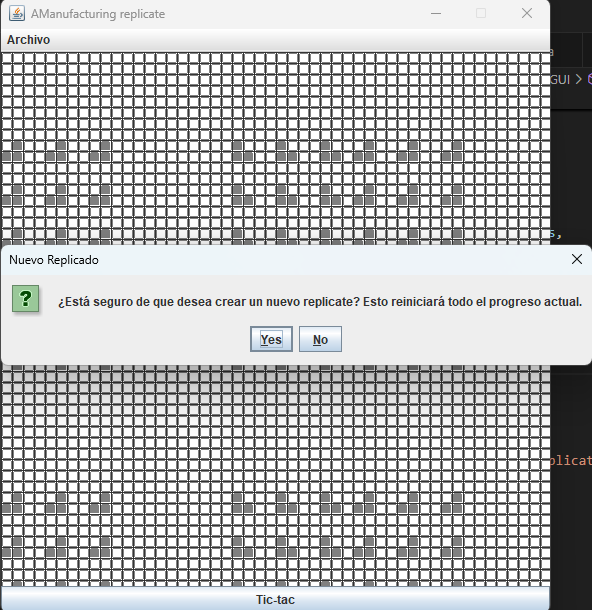


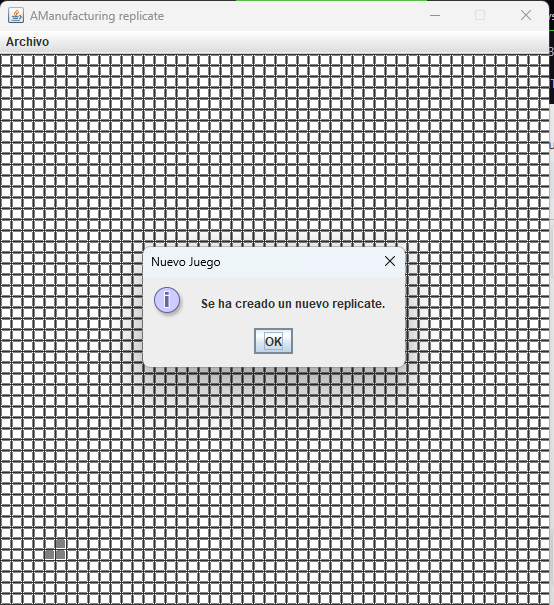


1. Construyan el método optionNew que crea un nuevo replicate. Capturen una pantalla significativa.

Se requiere ir a la capa de dominio para el funcionamiento correcto de este método (dominio es necesario dependiendo si el usuario quiere comenzar de nuevo replicate, entonces se crea otro objeto AManuFacturing y vuelve se repinta el ticTac) , así:







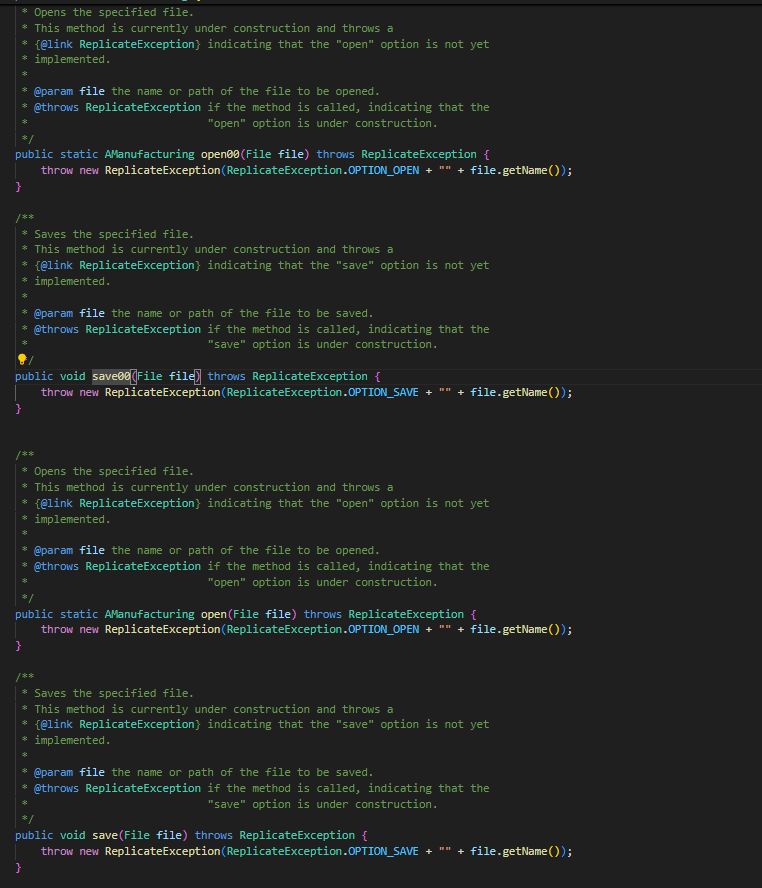
## Implementando salvar y abrir

**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

Las opciones salvar y abrir van a ofrecer servicios de persistencia del replicate como objeto. Los nombres de los archivos deben tener como extensión .dat.

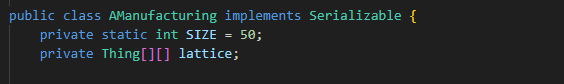
1. Copien las versiones actuales de open y save y renómbrenlos como open00 y save00.

Para hacer esto, solo es necesario copiar y pegar los dos mismos métodos de open y save de la fachada de dominio. Esto nos sirve para ver todas las versiones de estos métodos y ver como cada vez se implementan nuevas funcionalidades sobre los mismos métodos.

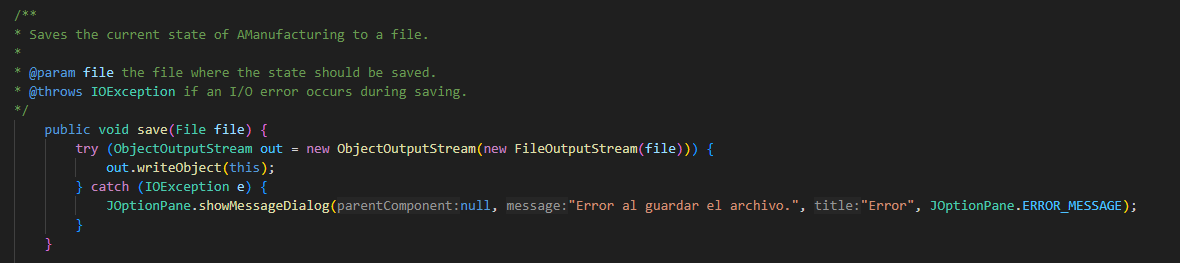


1. Construyan el método save que ofrece el servicio de guardar en un archivo el estado actual de replicate. Por ahora, para las excepciones solo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.

Es necesario ahora que AManuFacturing implemente la interfaz Serializable para poder guardar el estado de un objeto y poder dar con la implementación del método save.



En la fachada se atiende la excepción con IOException (mensaje de error general para el manejo de archivos).



En presentation, se hace con base al siguiente punto que es guardar el estado en onereplicate.dat.

|

|

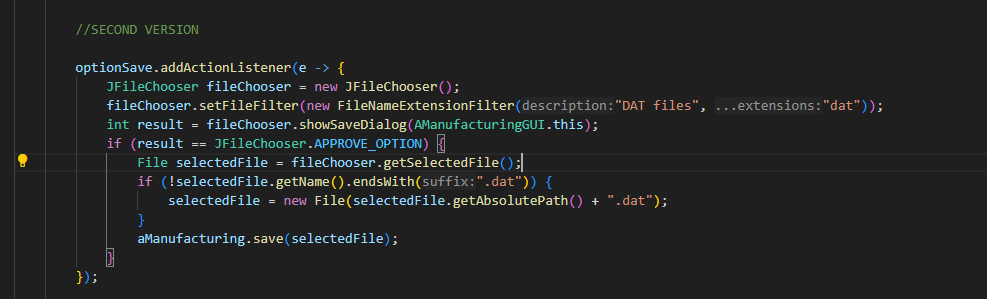
|

v

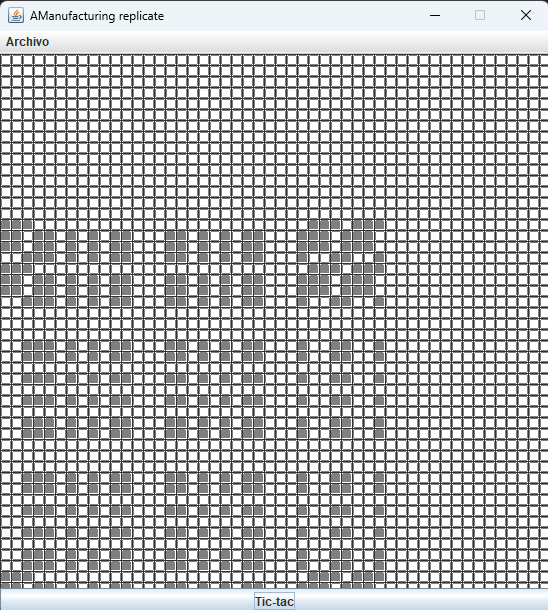
1. Validen este método guardando el estado obtenido después de dos clics como

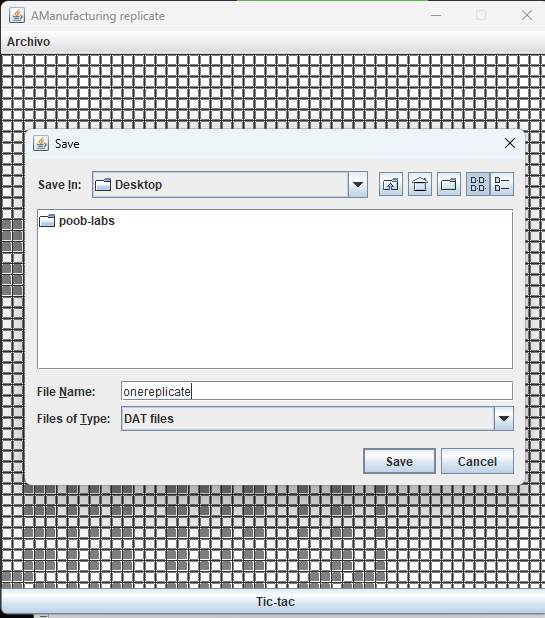
onereplicate.dat. ¿El archivo se creó en el disco? ¿Cuánto espacio ocupa?

Primero, en AManuFacturingGUI se crea con optionSave(controlador que maneja el guardar un archivo), el estado obtenido en onereplicate.dat.



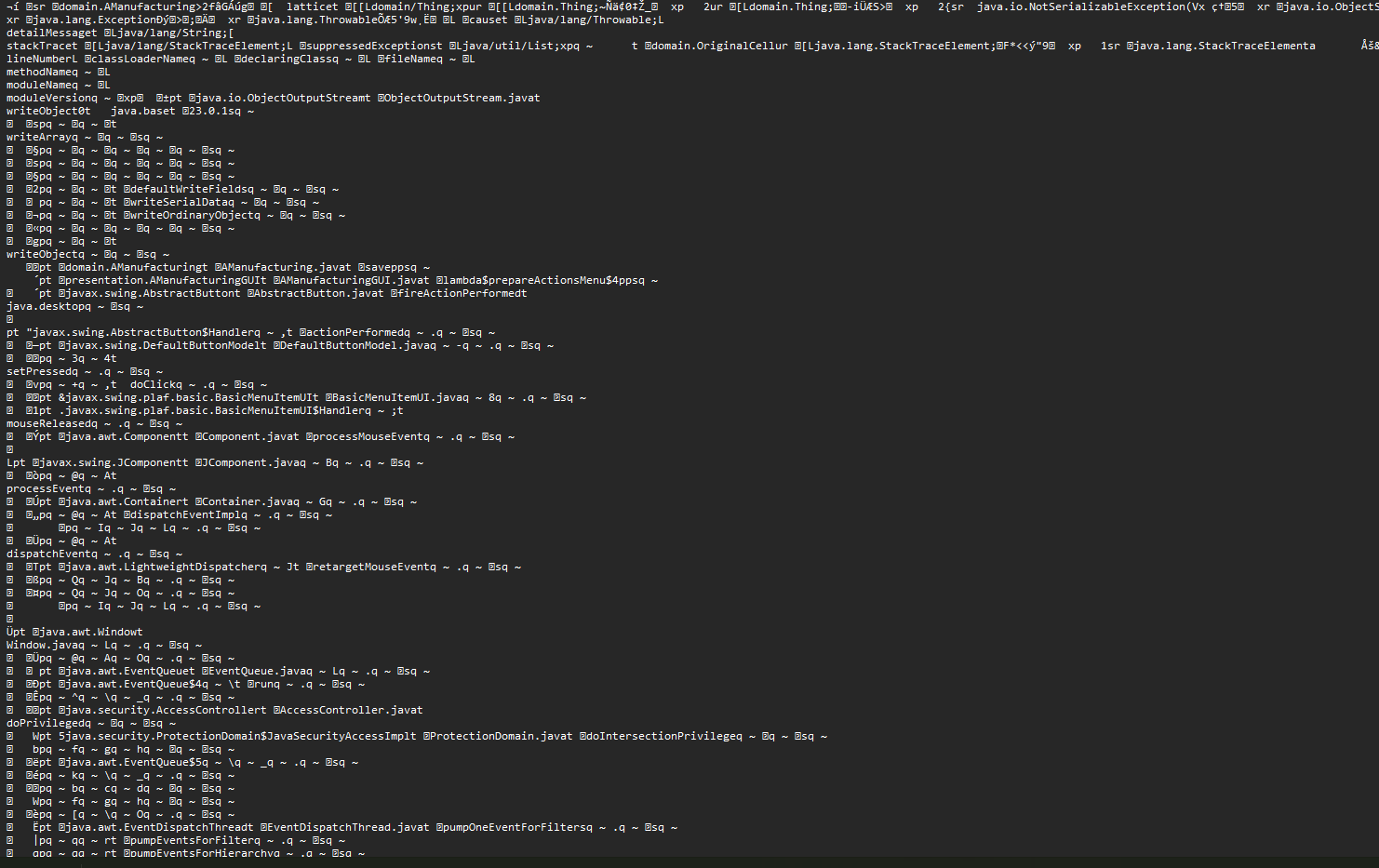
Con base al estado de este replicate se procede a guardarlo:





Ahora, si se hace doble clic sobre el estado de replicate al momento de guardarlo, se obtiene que el archivo **SI** se creó en el disco duro del computador en Desktop, y además ocupa un espacio de 4KB:

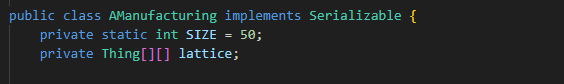




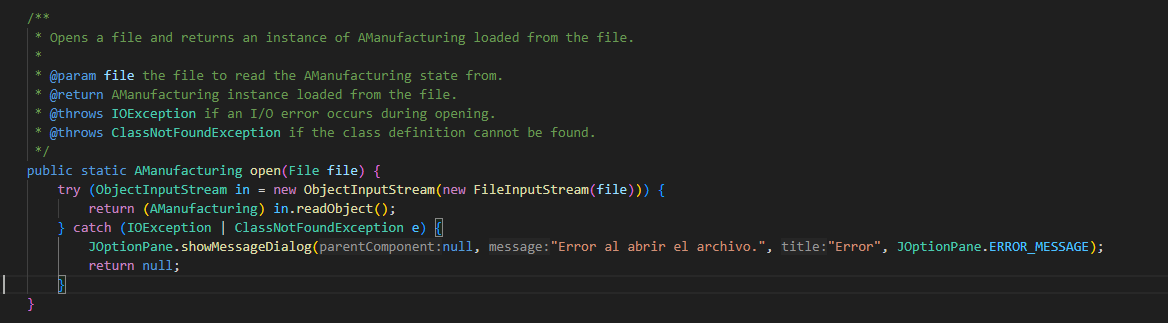
El archivo .dat es un “archivo de datos que almacena los datos de un programa en específico. La información del archivo solo es relevante para el programa que lo creó”.

1. Construyan el método open que ofrece el servicio de leer un replicate de un archivo. Por ahora, para las excepciones solo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.

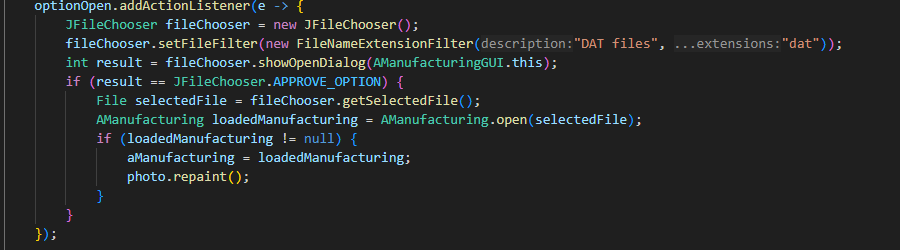
Es necesario ahora que AManuFacturing implemente la interfaz Serializable para poder leer el estado de un objeto y poder dar con la implementación del método open.



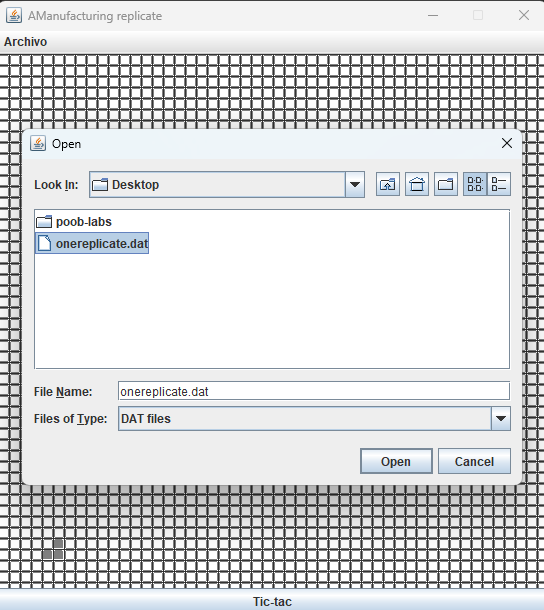
En la fachada se atiende la excepción con IOException (mensaje de error general para el manejo de archivos).

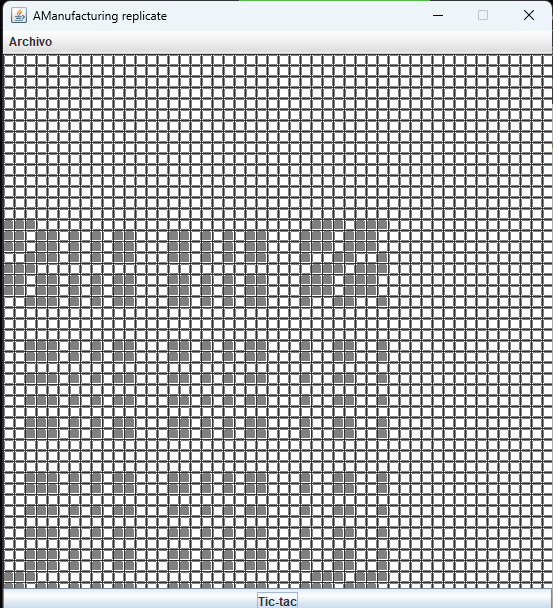


En presentation, se lee el estado de onereplicate.dat.(4KB)



Ahora, se procede a abrir onereplicate.dat para ver el estado que se guardó con el método save.



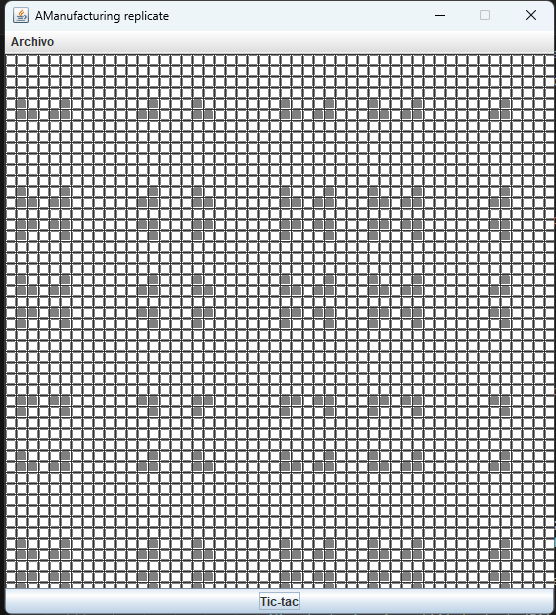


Se abrió el estado que estaba antes correctamente.

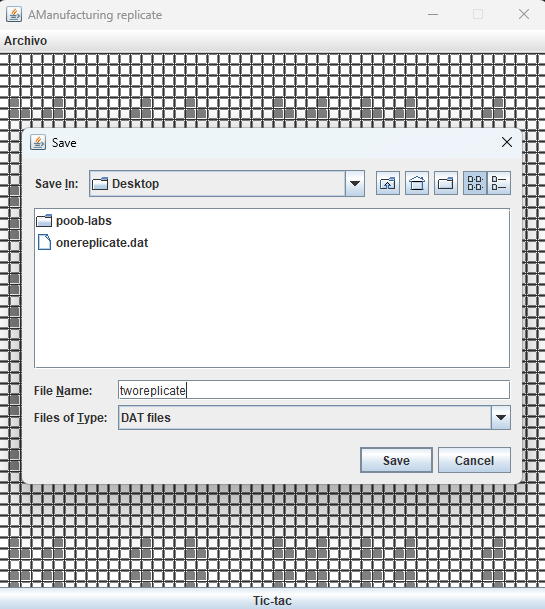
1. Realicen una prueba de aceptación para este método iniciando la aplicación, creando un nuevo estado y abriendo el archivo onereplicate.dat. Capturen imágenes significativas de estos resultados.

En los anteriores puntos, implícitamente ya se hizo una prueba de aceptación, pero vamos a hacer otra:

Entonces con este estado de juego:

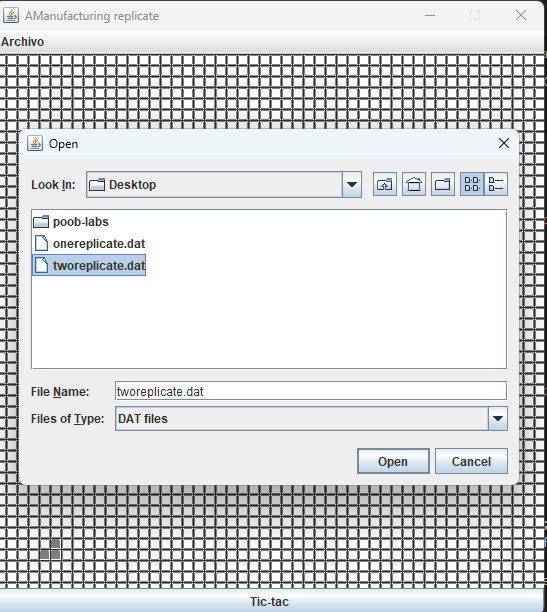


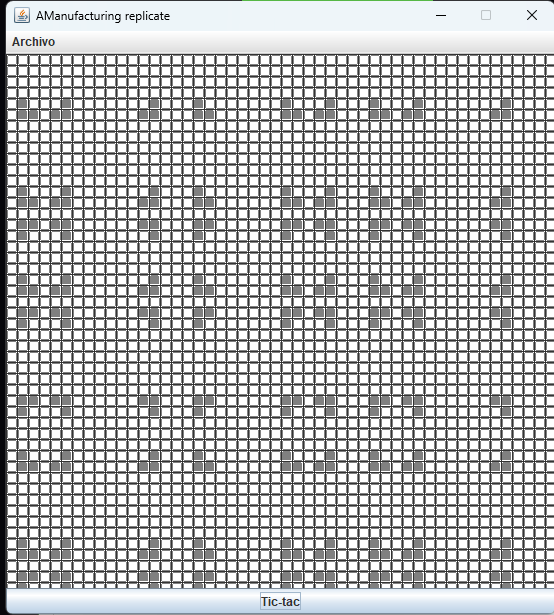
Se guarda el archivo como tworeplicate con extensión .dat





Ahora, procedo a abrir ese archivo y debería abrirme directamente el estado que guardé.





**Perfecto!**

## Implementando importar y exportar

**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

Estas operaciones nos van a permitir importar información del replicate desde un archivo de texto y exportarlo. Los nombres de los archivos de texto deben tener como extensión .txt

Los archivos texto tienen una línea de texto por cada elemento

En cada línea asociada un elemento se especifica el tipo y la posición. Cell 10 10

Poison 20 20

1. Copien las versiones actuales de import y export y renómbrenlos como import00 y

export00

Para hacer esto, solo es necesario copiar y pegar los dos mismos métodos de import y export de la fachada de dominio. Esto nos sirve para ver todas las versiones de estos métodos y ver como cada vez se implementan nuevas funcionalidades sobre los mismos métodos.

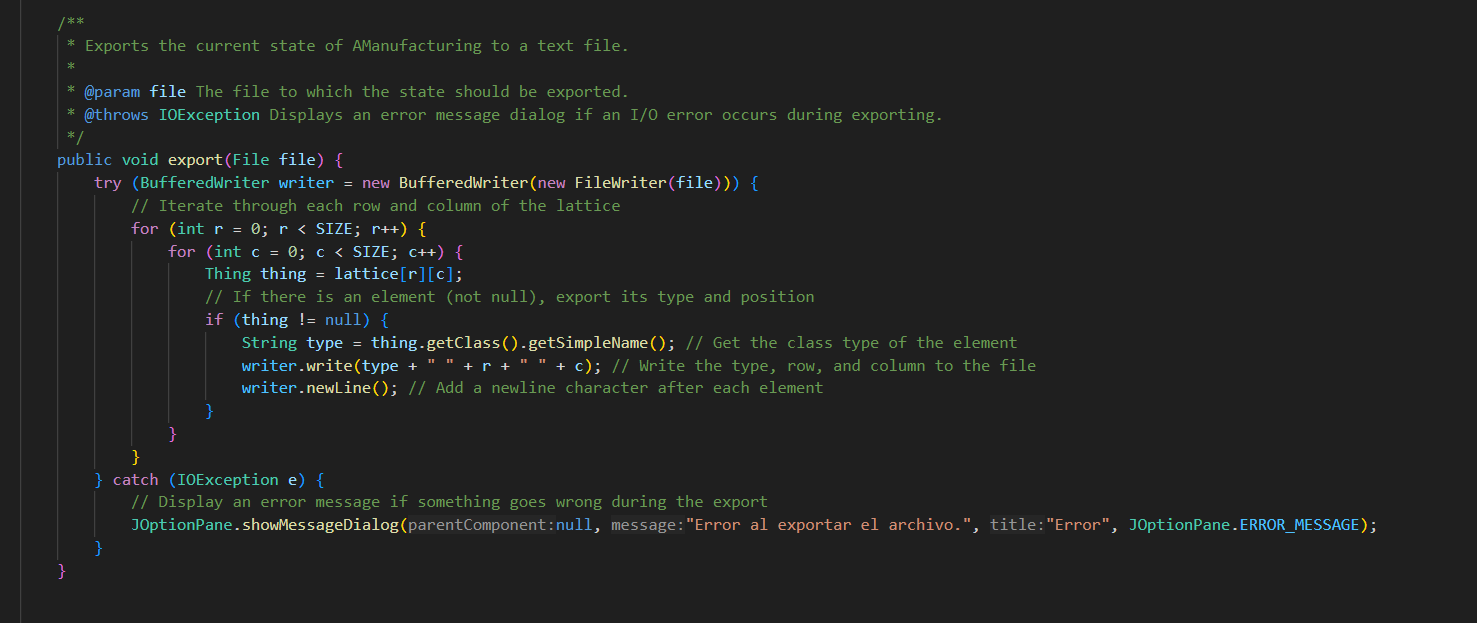


1. Construyan el método export que ofrece el servicio de exportar a un archivo texto, con el formato definido, el estado actual. Por ahora, para las excepciones solo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.

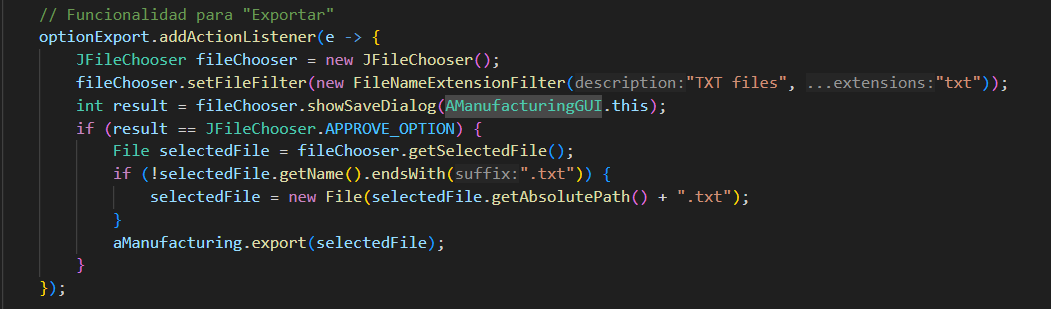
Creando el método con la funcionalidad solicitada y con la excepción en general IOException; entonces se procede a implementarlo tanto en GUI como en la fachada.

En GUI se establece el nuevo archivo con el que se va a exportar el estado de la partida (.txt) y en la fachada está la lógica para el funcionamiento de este.

**AManuFacturing**

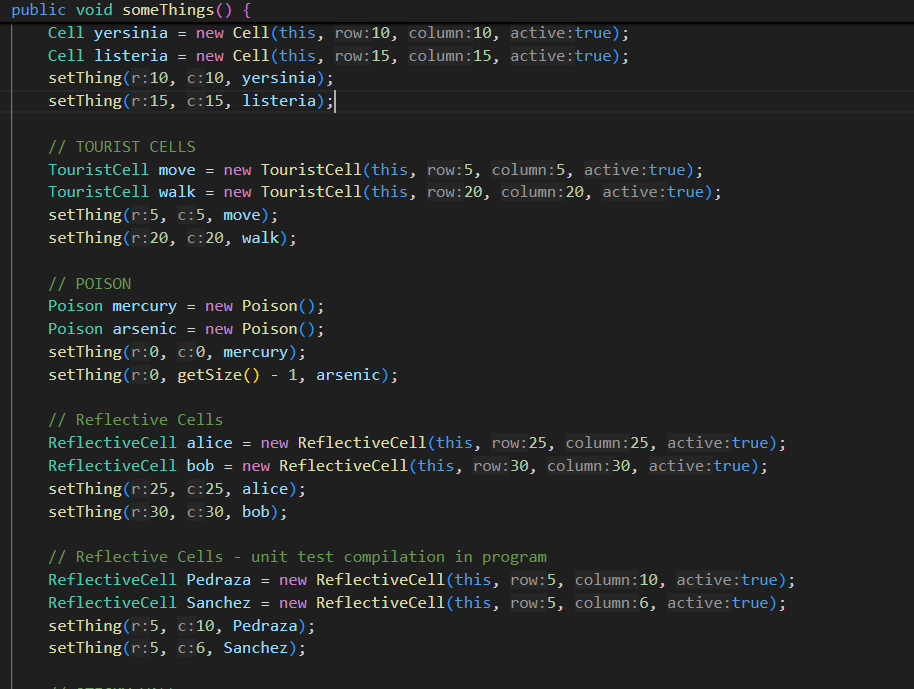
****

**AManuFacturingGUI**

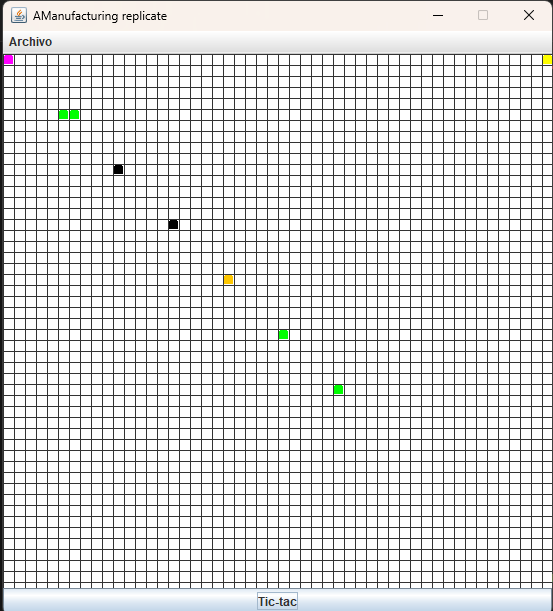
****

1. Realicen una prueba de aceptación de este método: iniciando la aplicación y exportando como onereplicate.txt. Editen el archivo y analicen los resultados. ¿Qué pasó?

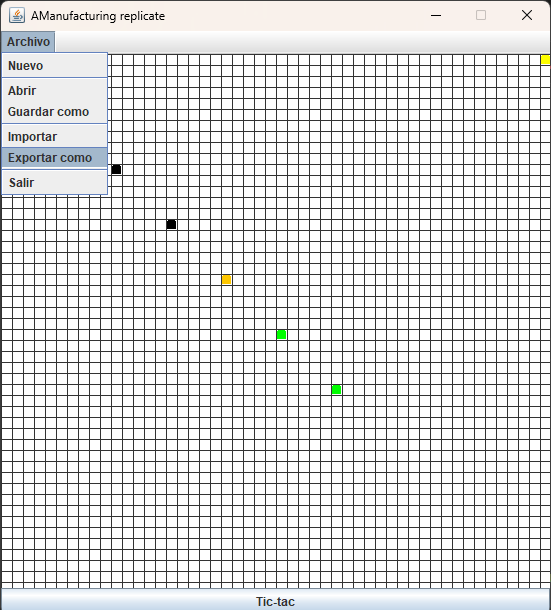
Teniendo de caso de prueba lo que está en el método de AManuFacturing someThings(), entonces al momento de exportar ese estado de las células como formato txt, entonces debe mostrar las posiciones en donde está cada tipo de célula.

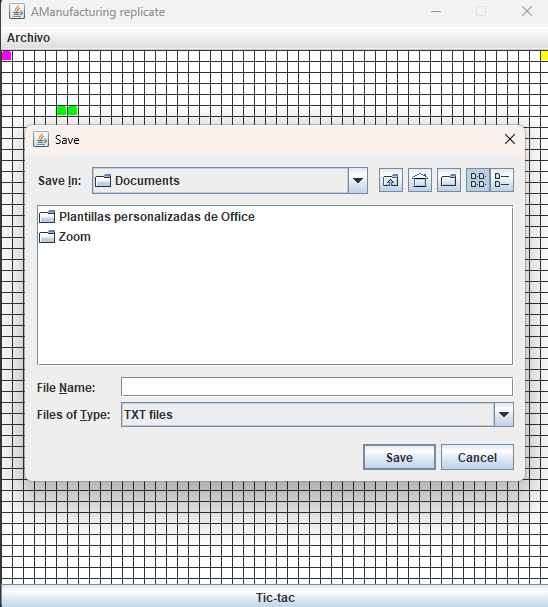


El estado actual de las células visualmente es:



Ahora se exportar como archivo txt:





Vemos que el archivo txt que generó es tal cual las posiciones en las que están ubicadas las células.



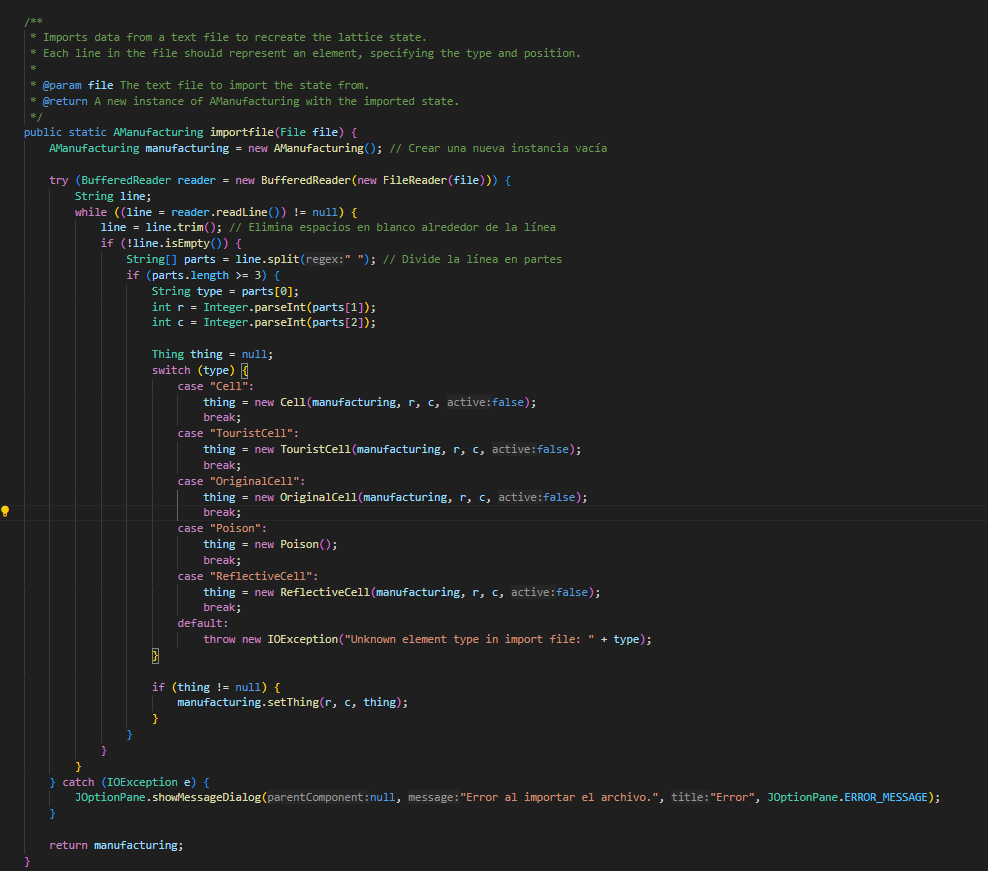
1. Construyan el método import que ofrece el servicio de importar de un archivo texto con el formato definido. Por ahora, solo considere un mensaje de error general. No olviden diseño y pruebas de unidad.

(Consulten en la clase String los métodos trim y split)

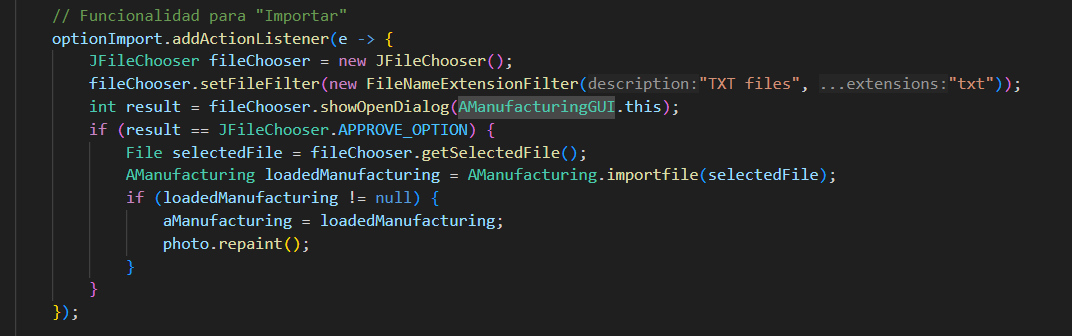
Con la misma lógica que se desarrolló en export, acá en import se implementa a su manera, para leer el archivo txt y de acuerdo a este cuando se importe ese archivo, entonces se crea la célula en la posición puesta en el archivo txt.

También teniendo en cuenta, el mensaje de error general y usando los métodos split y trim.

**AManuFacturing**

****

**AManuFacturingGUI**



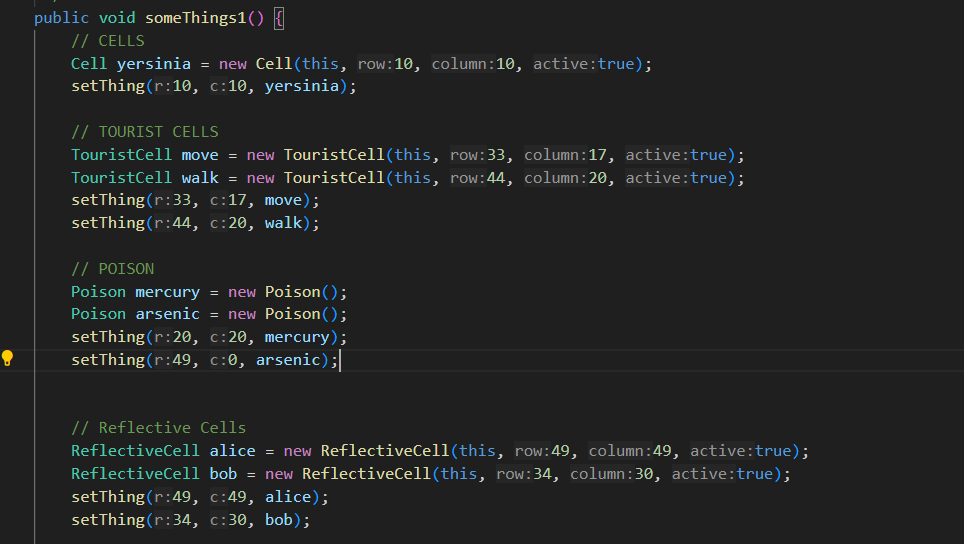
1. Realicen una prueba de aceptación de este par de métodos: iniciando la aplicación exportando a onereplicate.txt. saliendo, entrando, creando una nueva e importando el archivo otherreplicate.txt. ¿Qué resultado obtuvieron? Capturen la pantalla final.

Teniendo en cuenta los pasos solicitados, entonces primero compilo y creo varias células con diferente posición a las de las anteriores:

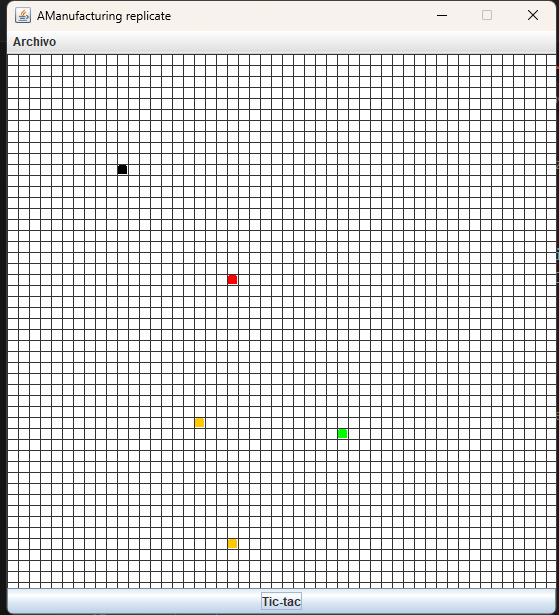
Con las células solicitadas al principio de este punto, y más células:

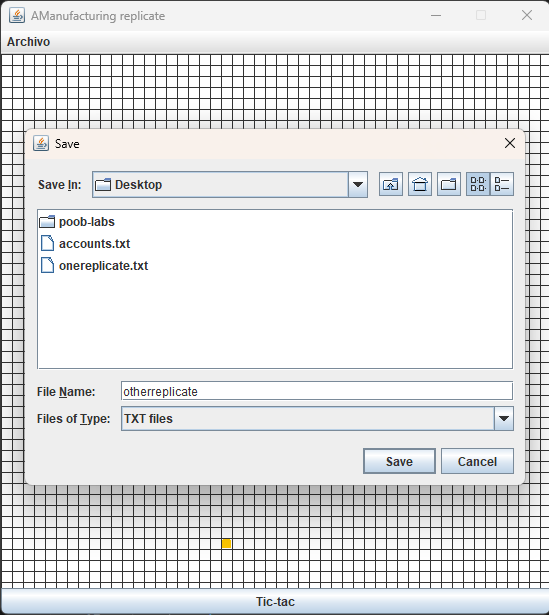
**Cell 1010**

**Poison 20 20**

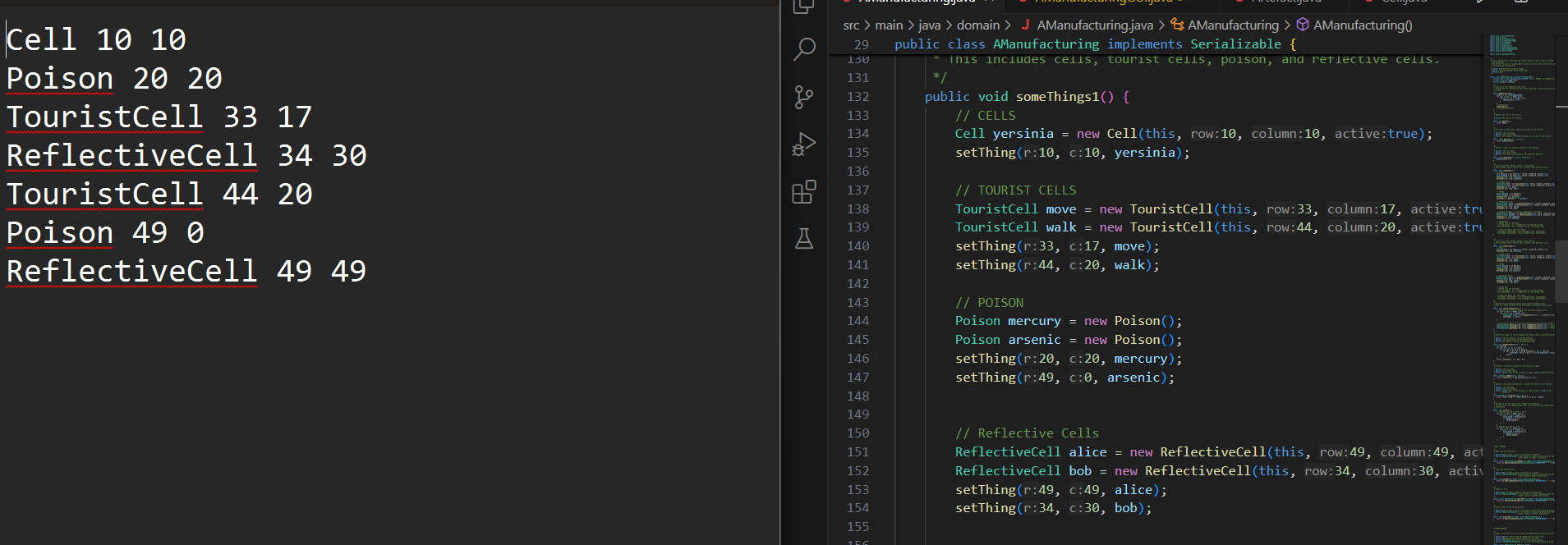


Ahora se ejecuta y se exporta como archivo txt con el nombre de otherreplicate:



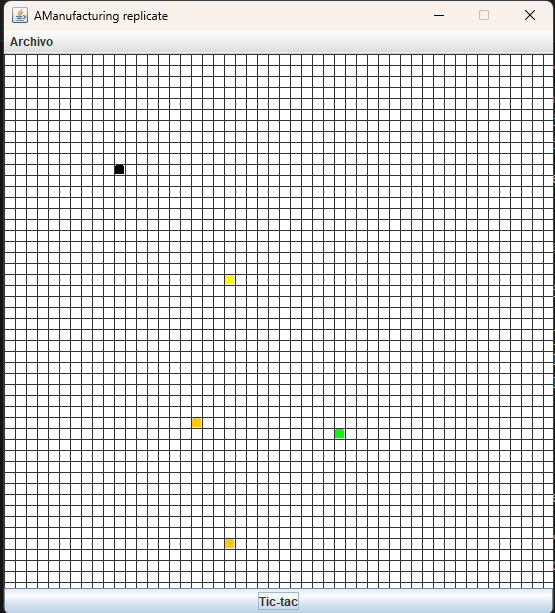


Vemos que la exportación fue exitosa:



Ahora, vamos a mirar cuando importamos ese archivo txt debe pintar las células en las posiciones adecuadas:



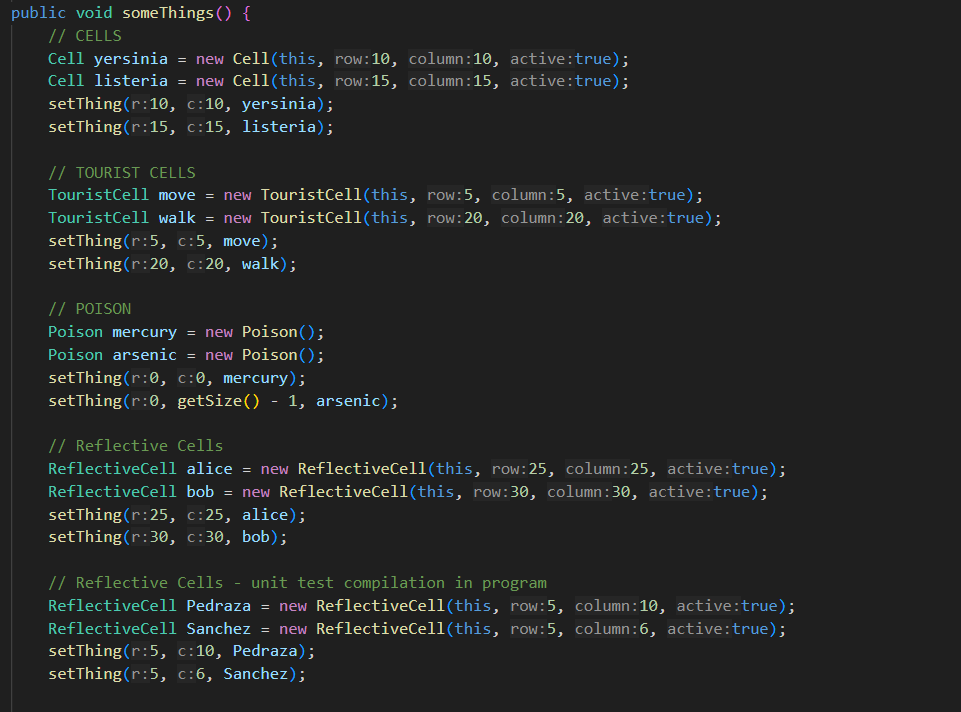


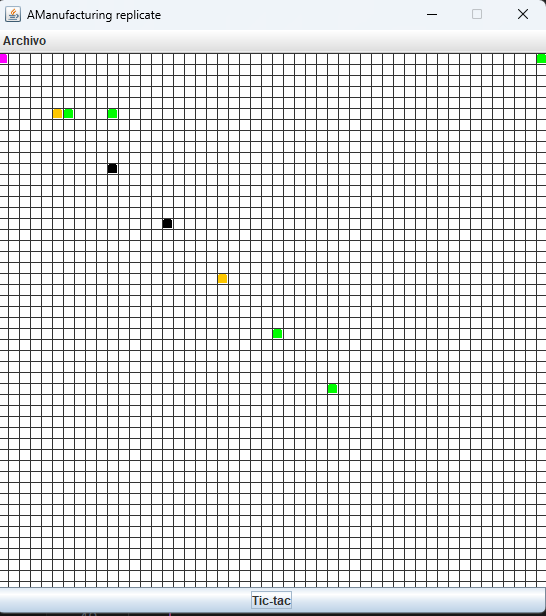
Todo es correcto, vemos que Poison cambia de color, pero ya es por su definición de que siempre cambia de color cada vez que se instancia, y como se está importando sin guardar colores, entonces por eso cambia, de resto, las otras células están igual.

1. Realicen otra prueba de aceptación de este método escribiendo un archivo de texto correcto en onereplicate.txt. e importe este archivo. ¿Qué resultado obtuvieron? Capturen la pantalla.

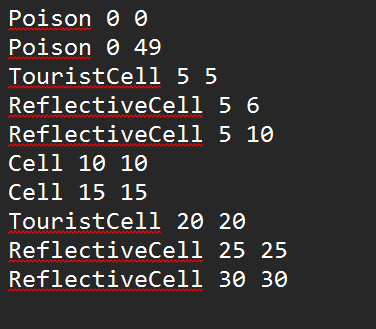
Ahora, el último caso de aceptación implica que se exporta cualquier estado de las células que se hayan creado, y luego además de importar ese archivo txt, se modifica antes ese archivo, creando más tipos de células, pero desde el notepad.

Y con base al requisito, es teniendo en cuenta onereplicate.txt, que este se basó en someThings(), así:





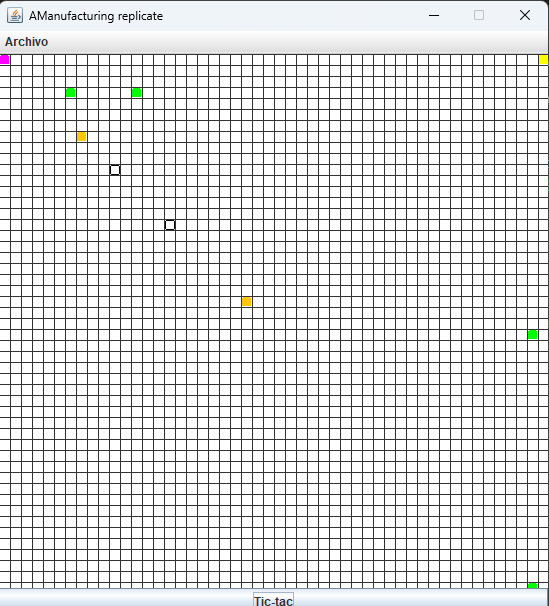


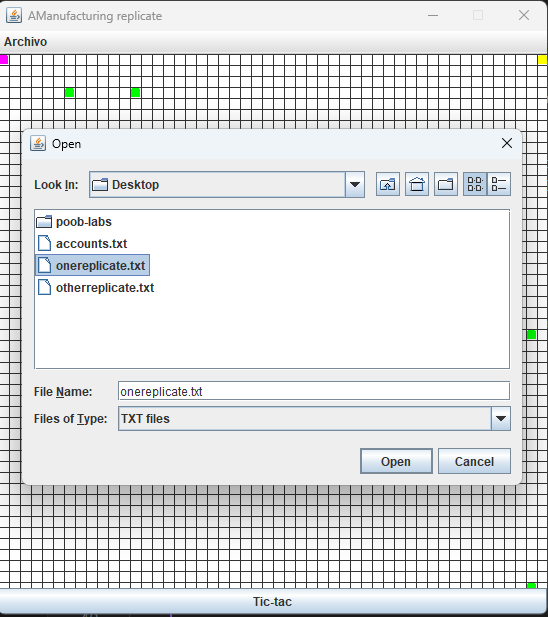


Ahora, si yo añado una Cell (34 34) y Poison (30,0), cuando se importe onereplicate.txt se debe actualizar el gridLayout con las nuevas células, más las anteriores.

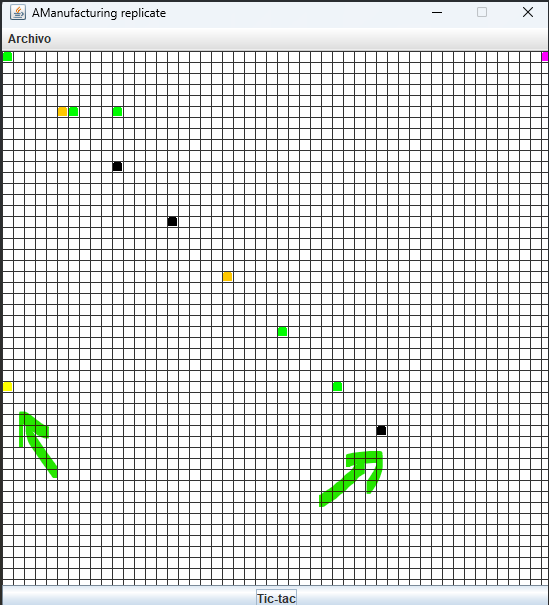


Se hizo tic tac con la configuración inicial para verificar que cuando se import, photo se repinta con lo que se importe





Vemos que cambiando el archivo txt agregando las dos nuevas células, entonces se modificó cuando se importó el nuevo archivo.



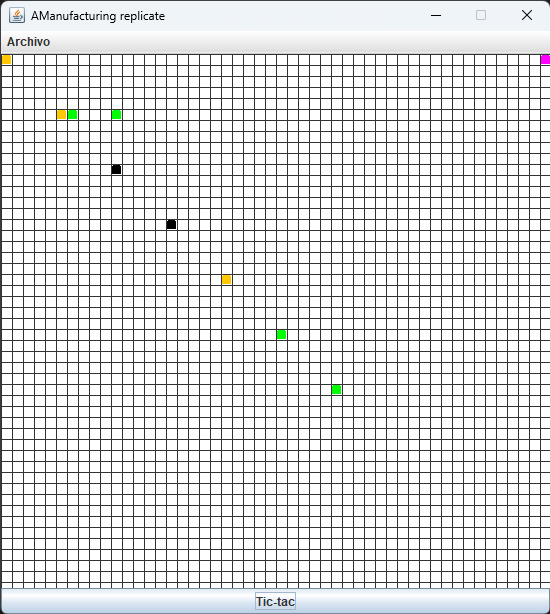
## Analizando comportamiento

**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

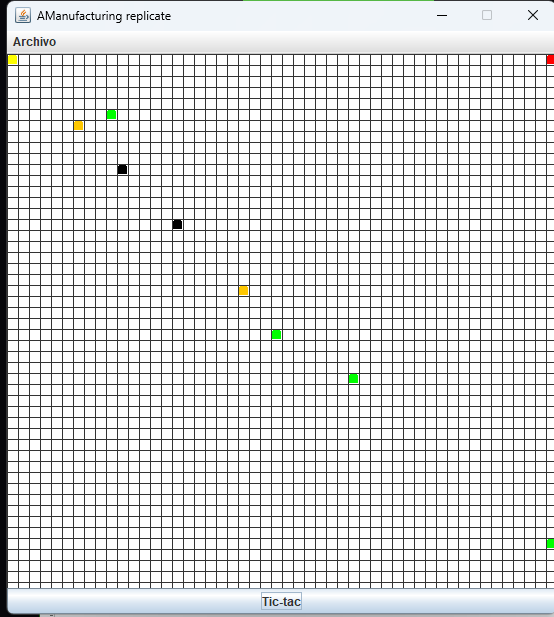
1. Ejecuten la aplicación, den tres clics, salven a un archivo cualquiera y ábranlo. Describan el comportamiento.

Teniendo en cuenta someThings, se procede a hacer 3 ticTac

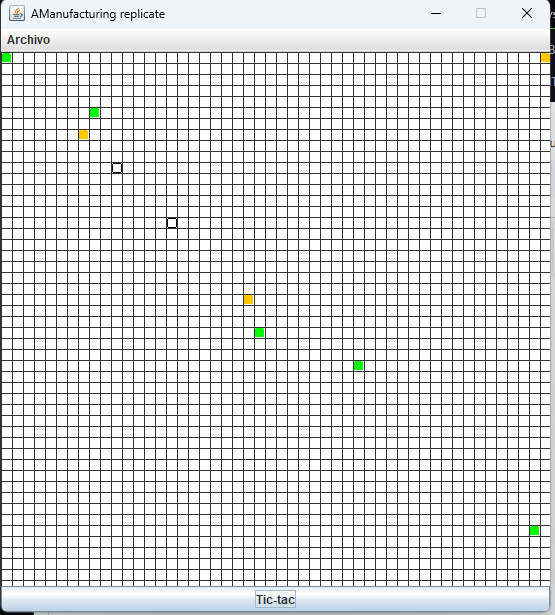
**CONFIGURACIÓN INICIAL**

****

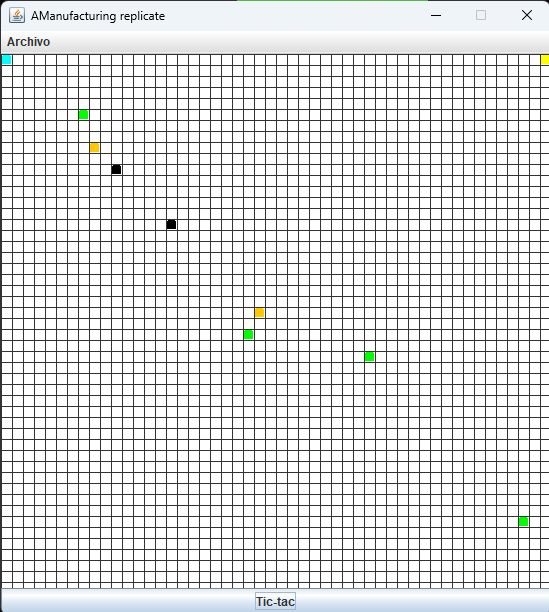
**1 TICTAC**



2 TICTAC

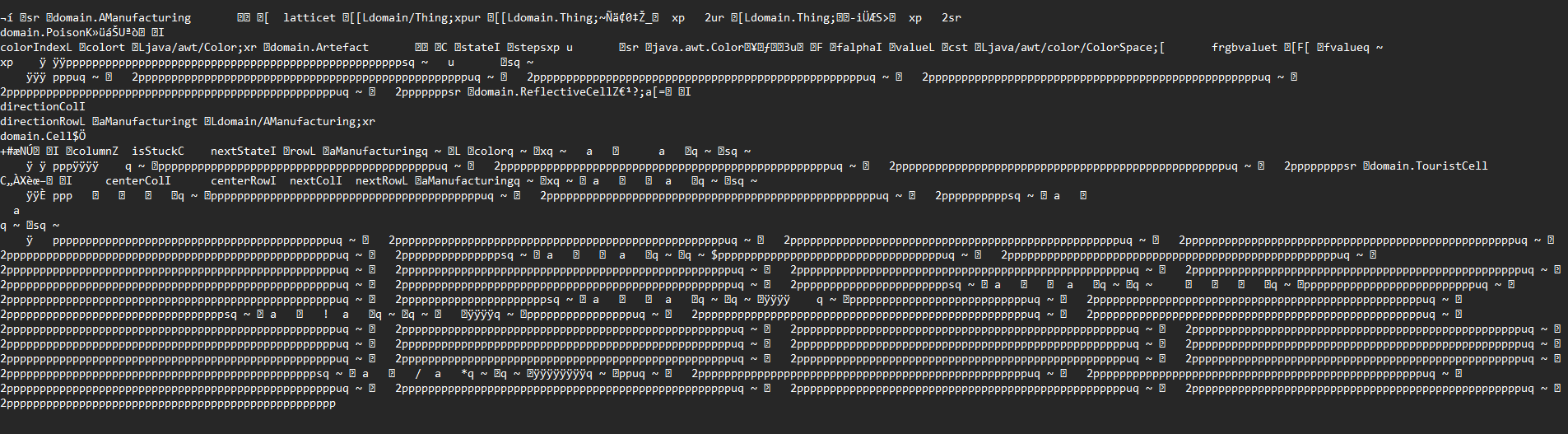


3 TICTAC



Ahora, se guarda el estado de ese archivo en formato .dat



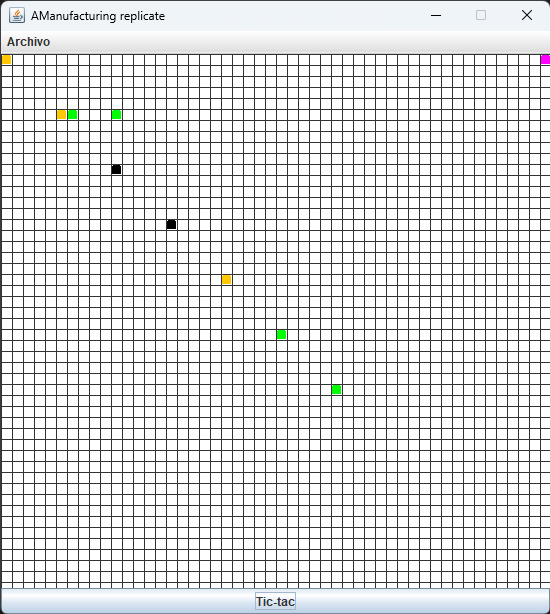


El comportamiento de guardar un archivo es muy extraño porque se guarda el estado del archivo en formato que solo la máquina la entiende cuando se va a abrir el .dat.

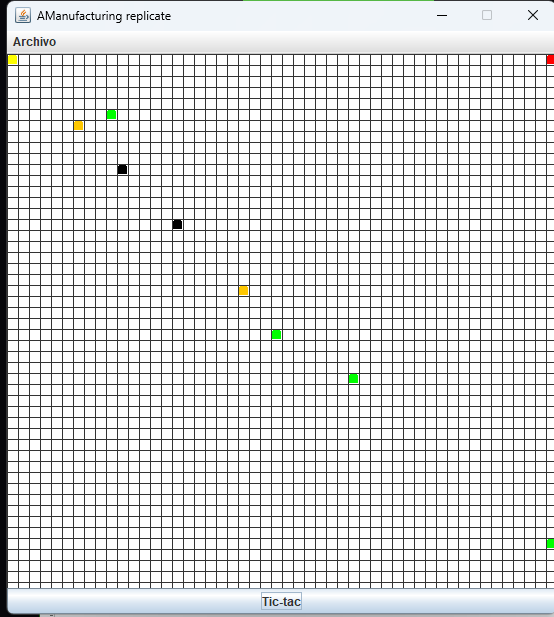
1. Ejecuten la aplicación, tres clics, exporten a un archivo cualquiera e importen. Describan el comportamiento.

Teniendo en cuenta la misma configuración y los mismos 3 ticTac, ahora se exporta como .txt y luego se importa ese archivo.

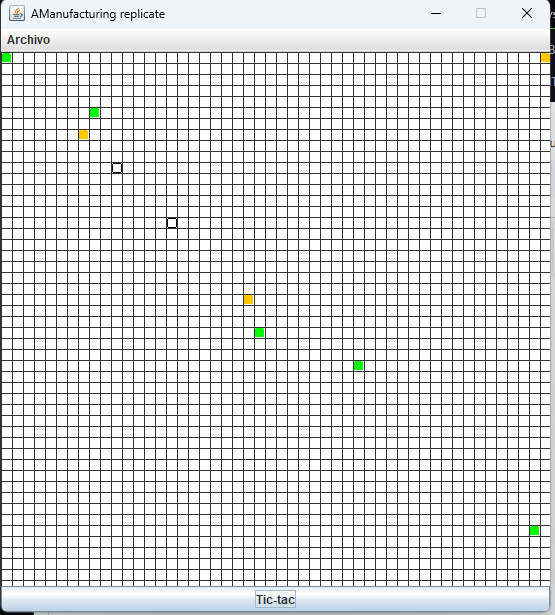
**CONFIGURACIÓN INICIAL**

****

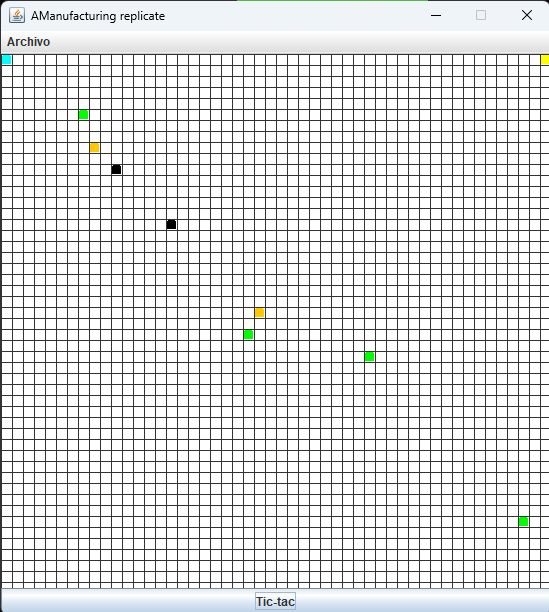
**1 TICTAC**



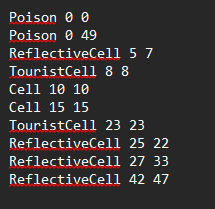
2 TICTAC



3 TICTAC

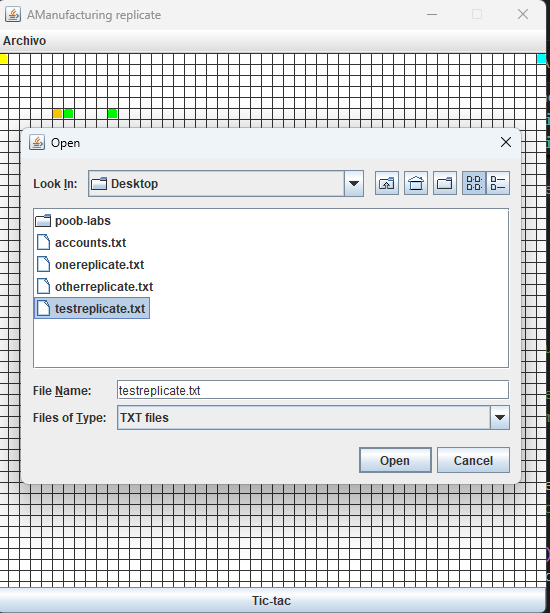


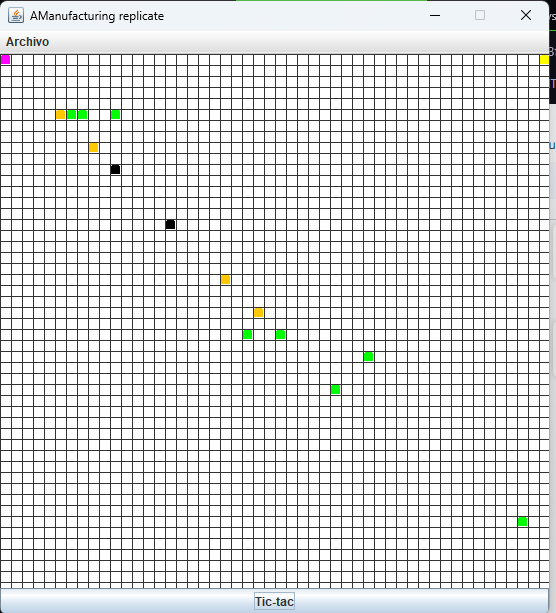




El comportamiento es más limpio en formato .txt porque se evidencia claramente las posiciones de cada célula de manera organizada, cada una en distinta línea.

Si se importa, debe cargar esa configuración que se exportó al hacer ticTac.

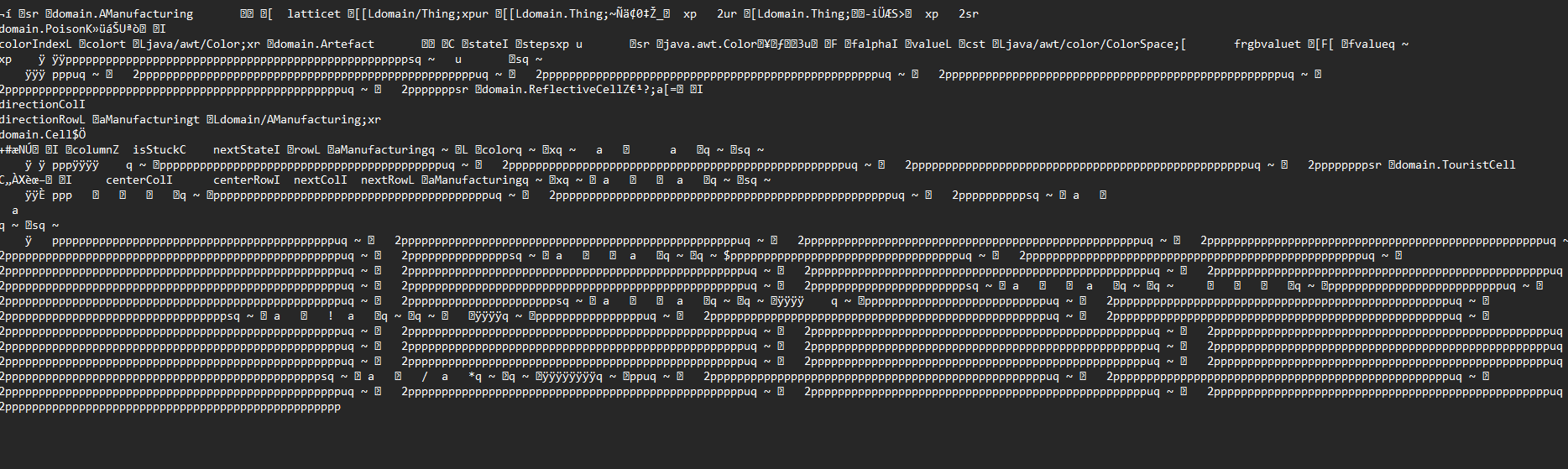


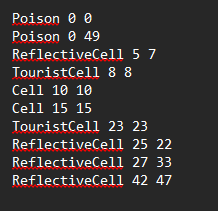


1. ¿Qué diferencias ven el comportamiento 1. y 2.? Expliquen los resultados.

Las diferencias resaltan en como se ven los datos cuando se guarda o cuando se exporta un archivo, mientras que save tiene los datos de una forma que no es posible leer de manera rápida; export si muestra los datos de manera organizada y clara.

También, nos damos cuenta de que esta diferencia se da por el formato con el que se guarda/exporta, ya sea, .dat o .txt. Además, estos dos métodos cumplen por definición dos funcionalidades distintas.



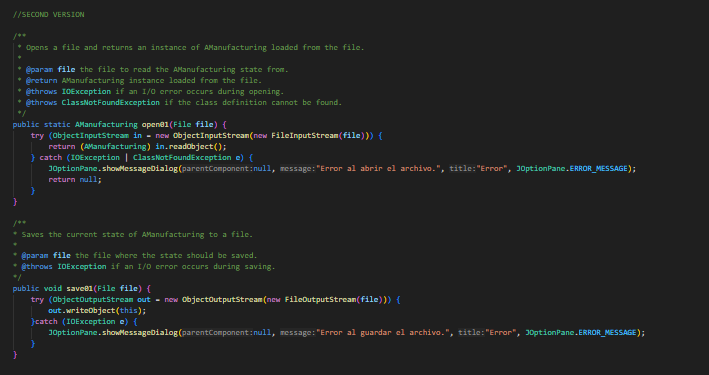


## Perfeccionando salvar y abrir

**[En lab06.doc, \*.asta y \*.java] [NO OLVIDEN BDD y MDD]**

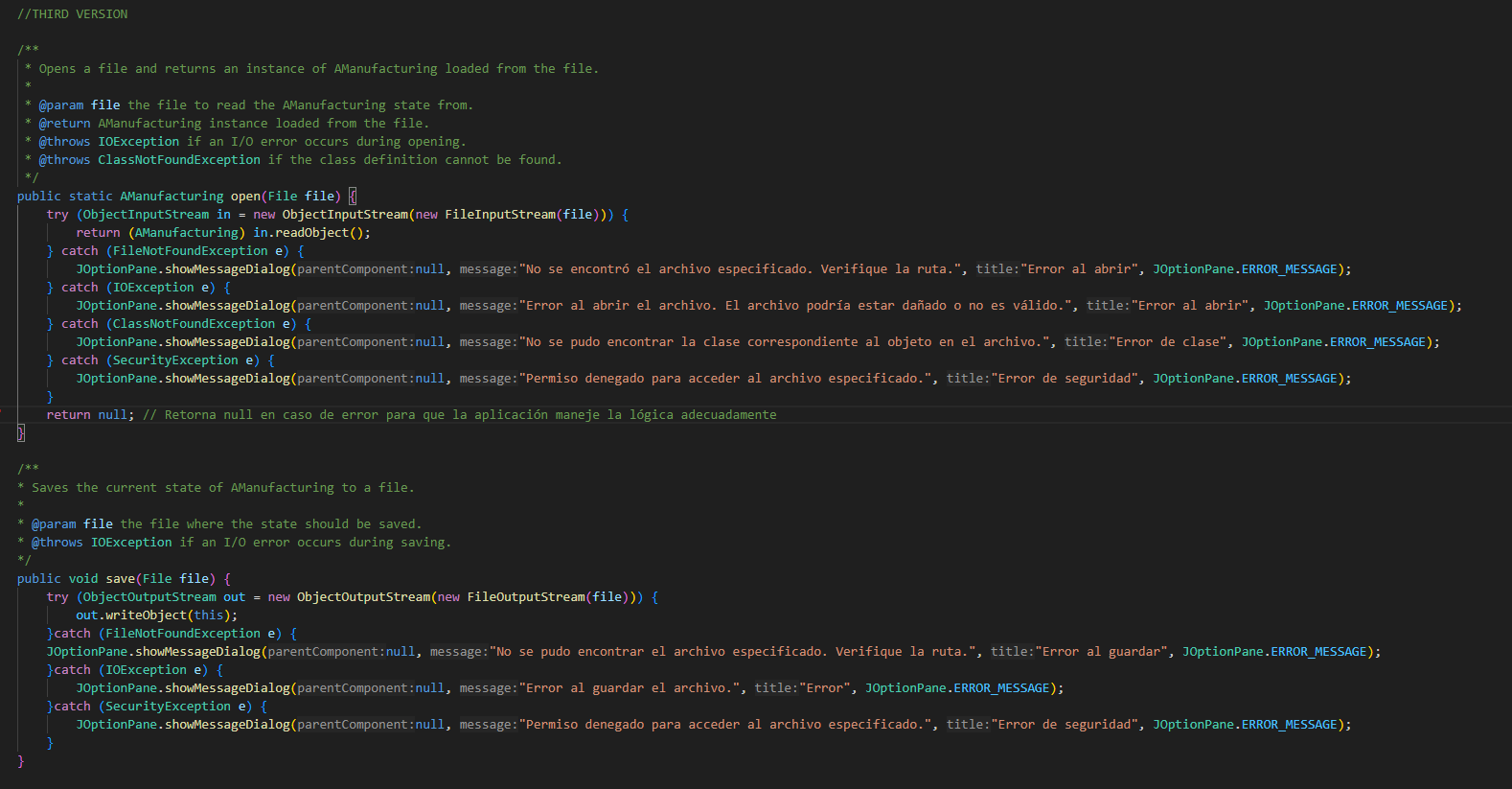
1. Copien las versiones actuales de open y save y renómbrenlos como open01 y save01

Creando la última versión de las funcionalidades de estas opciones, entonces se hace lo solicitado.



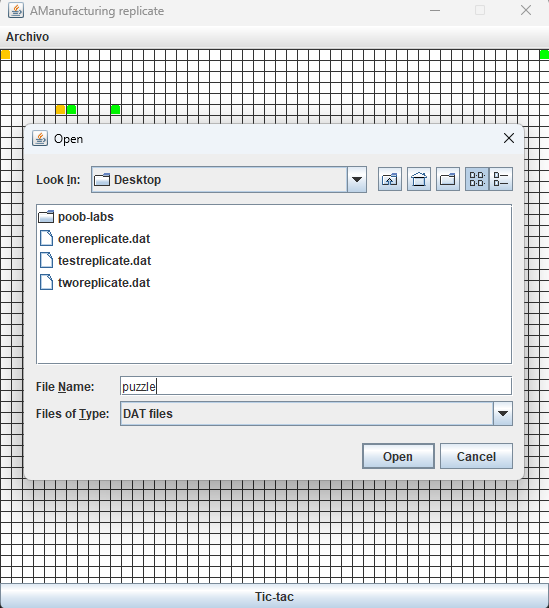
1. Perfeccionen el manejo de excepciones de los métodos open y save detallando los errores. No olviden pruebas de unidad.

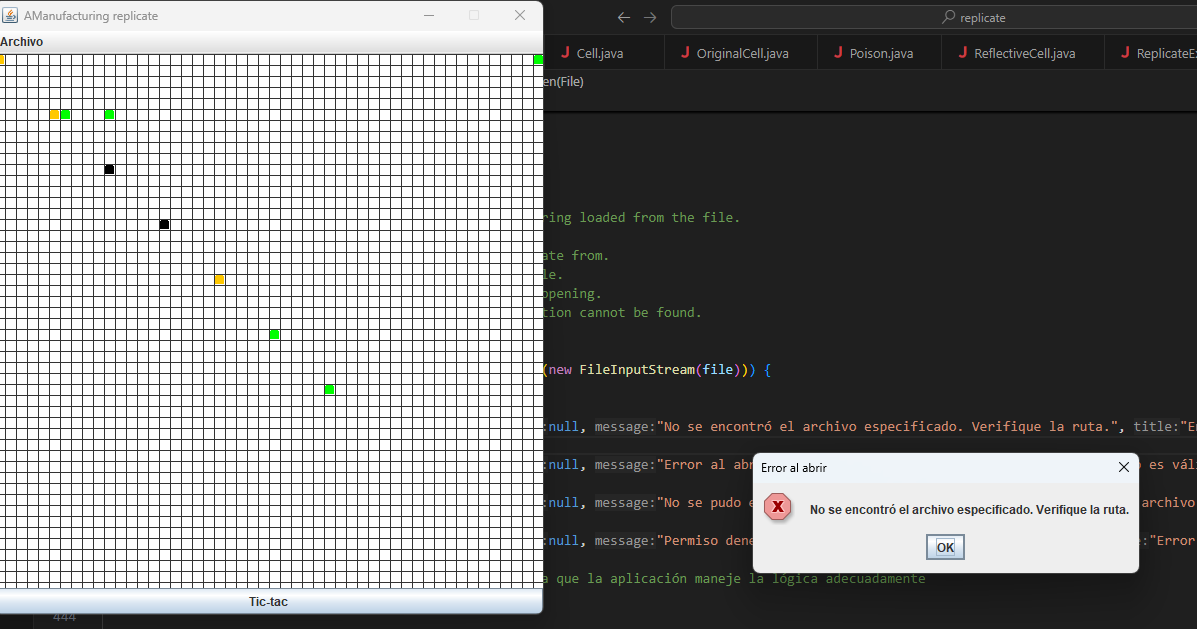
Se añaden excepciones más detalladas del sistema, en cuanto a seguridad, archivo no encontrado, IOException, clase no encontrada; solo en AManufacturing



1. Realicen una prueba de aceptación para validar **uno** de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.

Para validar un caso de que el archivo no está encontrado, entonces vamos a abrir un archivo que tiene de nombre puzzle.dat, es algo que no está en el contexto de replicate, así que debe mandar el mensaje de excepción FileNotFound.





## Perfeccionando importar y exportar.

**[En lab06.doc, \*.asta , replicateErr.txt \*.java] [NO OLVIDEN BDD y MDD]**

1. Copien las versiones actuales de import y export y renómbrenlos como import01 y

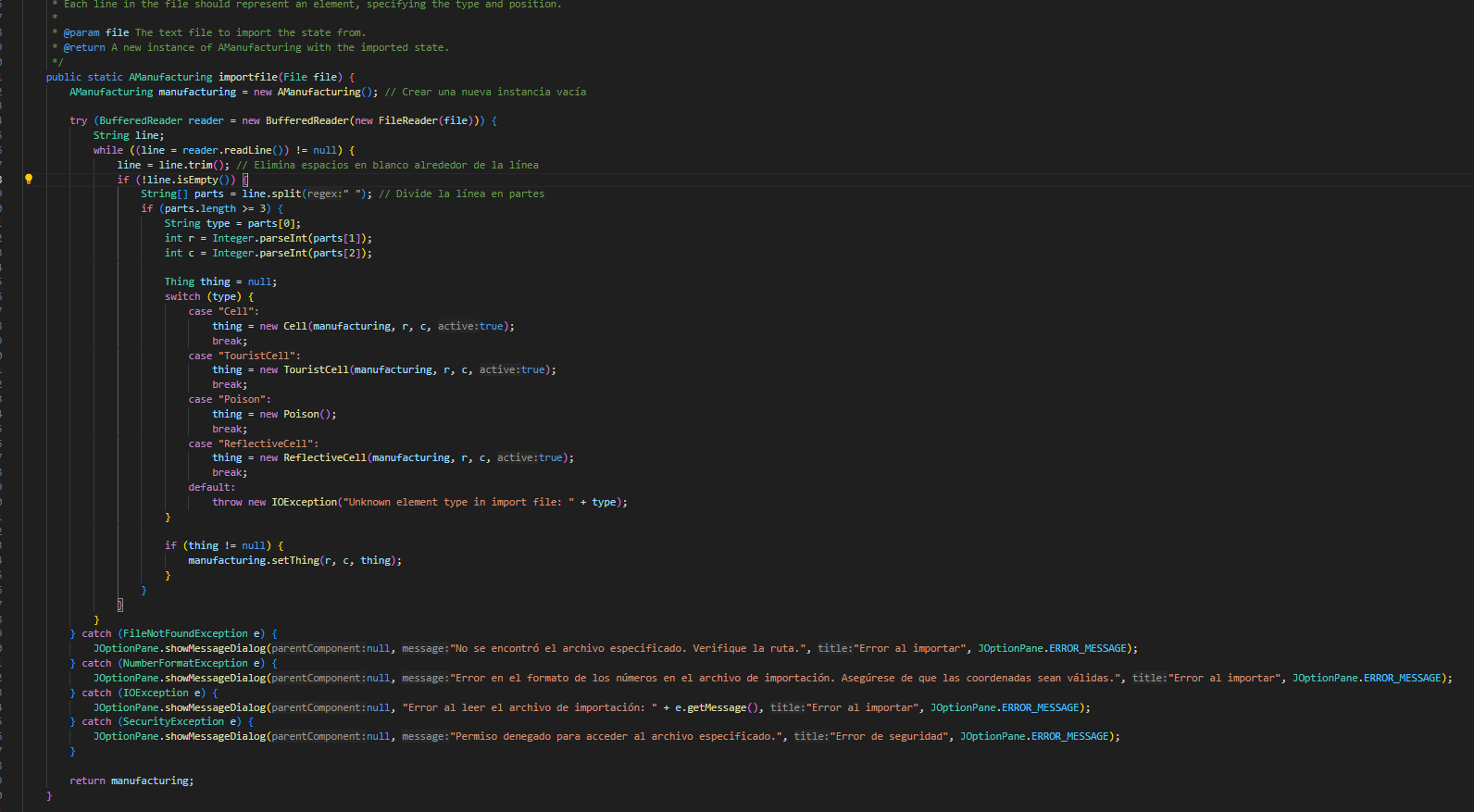
export01

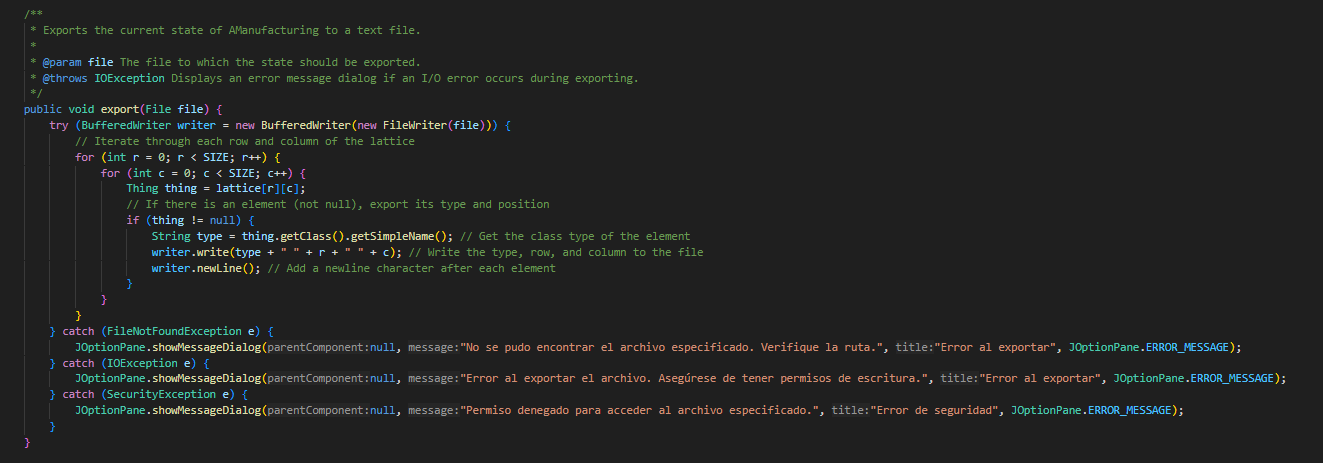
Creando la última versión de las funcionalidades de estas opciones, entonces se hace lo solicitado.



1. Perfeccionen el manejo de excepciones de los métodos import y export detallando los errores. No olviden pruebas de unidad.

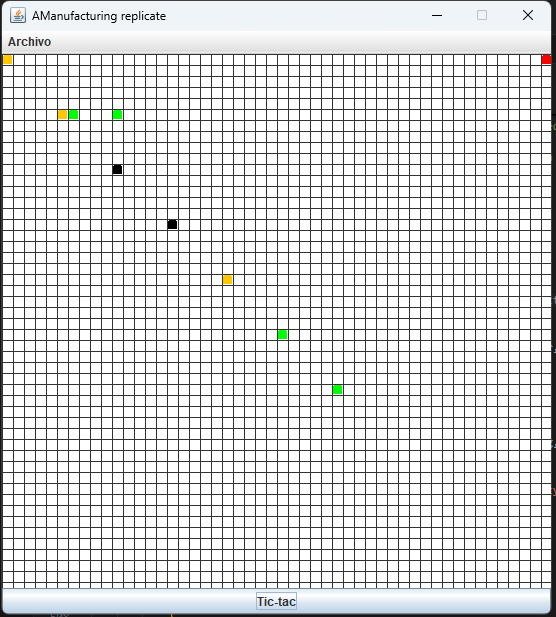
Se añaden excepciones más detalladas del sistema, en cuanto a seguridad, archivo no encontrado, númeroNoVálido, IOException, ArrayIndexOutOfBoundsException Solo en AManufacturing.

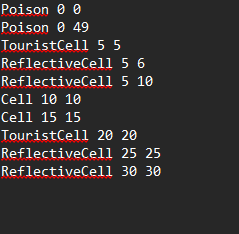




1. Realicen una prueba de aceptación para validar **uno** de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.

Se va a validar con la excepción, ArrayIndexOutOfBoundsException, con el método de exportar cualquier configuración.

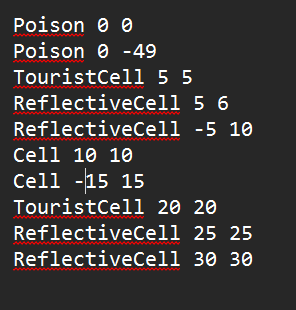




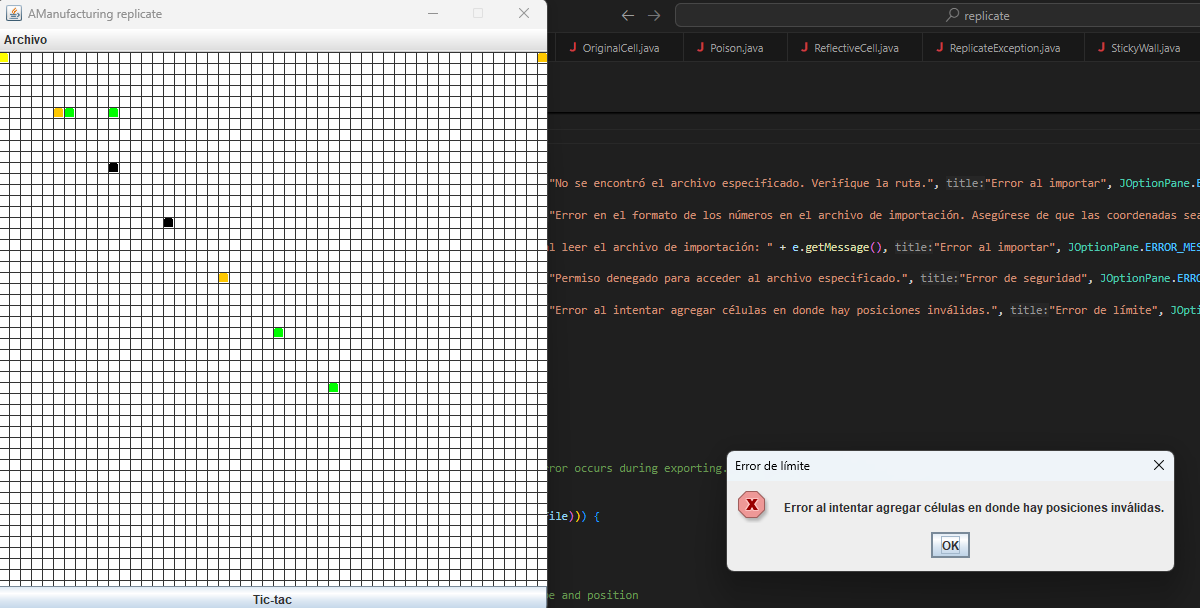
Ahora, ya que se exportó ese archivo llamado acceptanceTestImport



Entonces, ahora voy a modificar ese archivo y poniendo posiciones negativas para que al momento de intentar importar ese archivo txt me genera el mensaje de excepción.



Con solo que haya uno malo, no crea las otras células



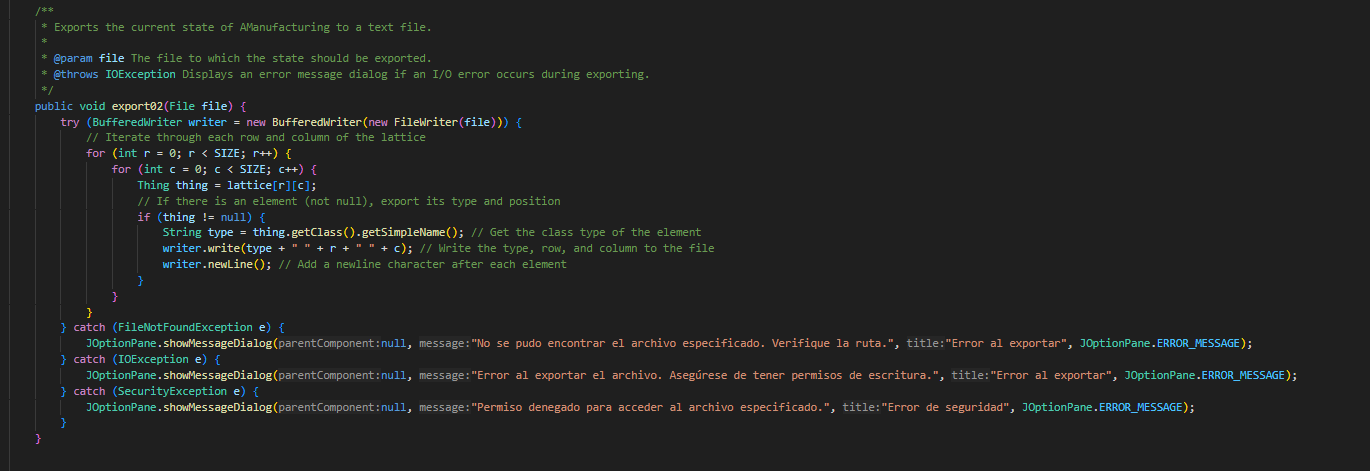
## Perfeccionando importar. Hacia un minicompilador.

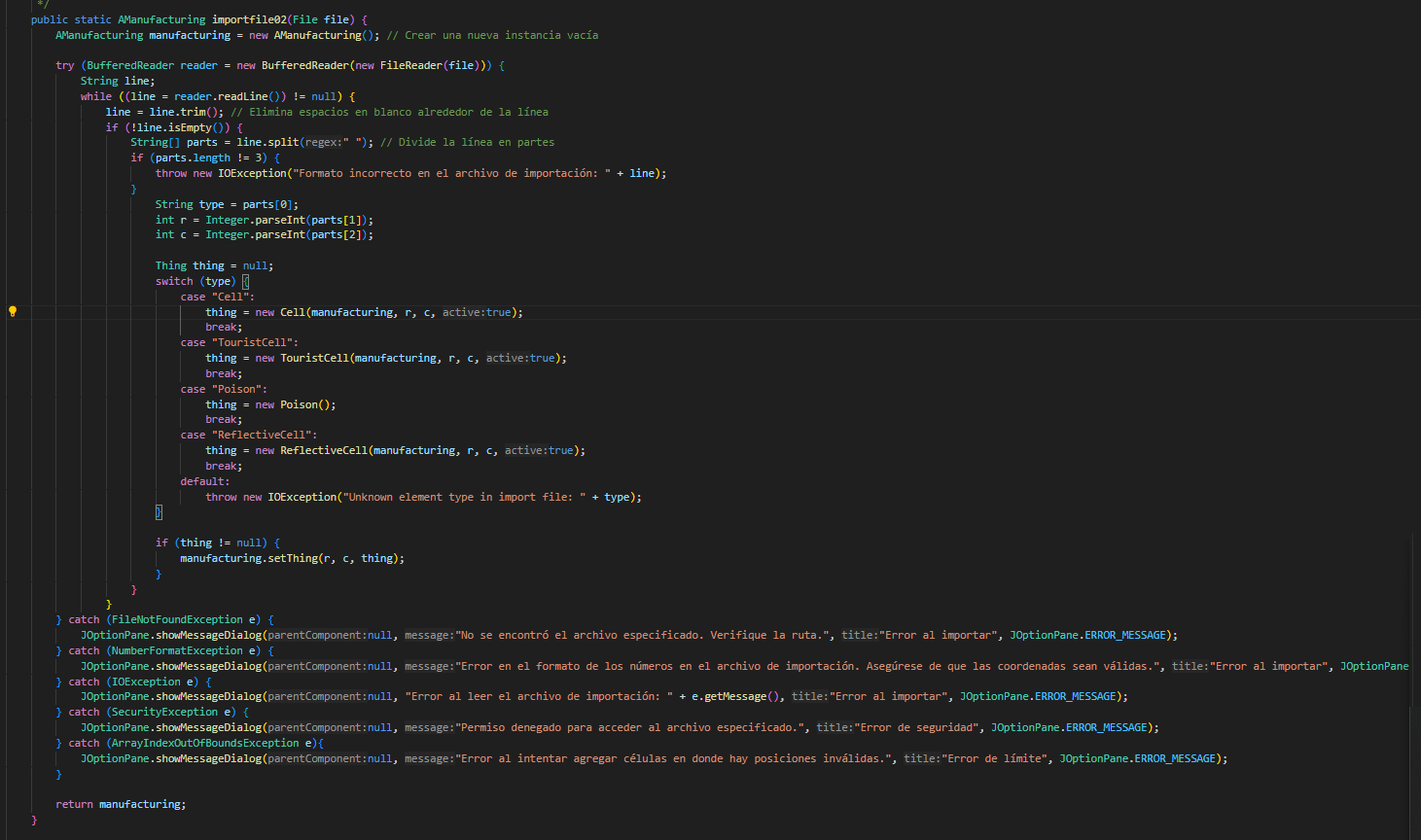
**[En lab06.doc, \*.asta , replicateErr.txt \*.java] [NO OLVIDEN BDD y MDD]**

1. Copien las versiones actuales de import y export y renómbrenlos como import02 y

export02

Cumpliendo con lo solicitado:





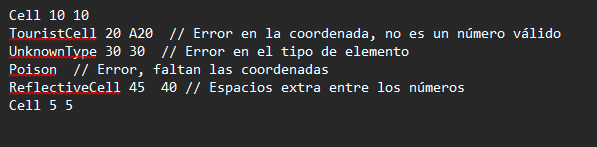
1. Perfeccionen el método **import** para que, además de los errores generales, en las excepciones indique el detalle de los errores encontrados en el archivo (como un compilador) : número de línea donde se encontró el error, palabra que tiene el error y causa de error.

Ahora, cumpliendo con estos requerimientos, se obtiene:



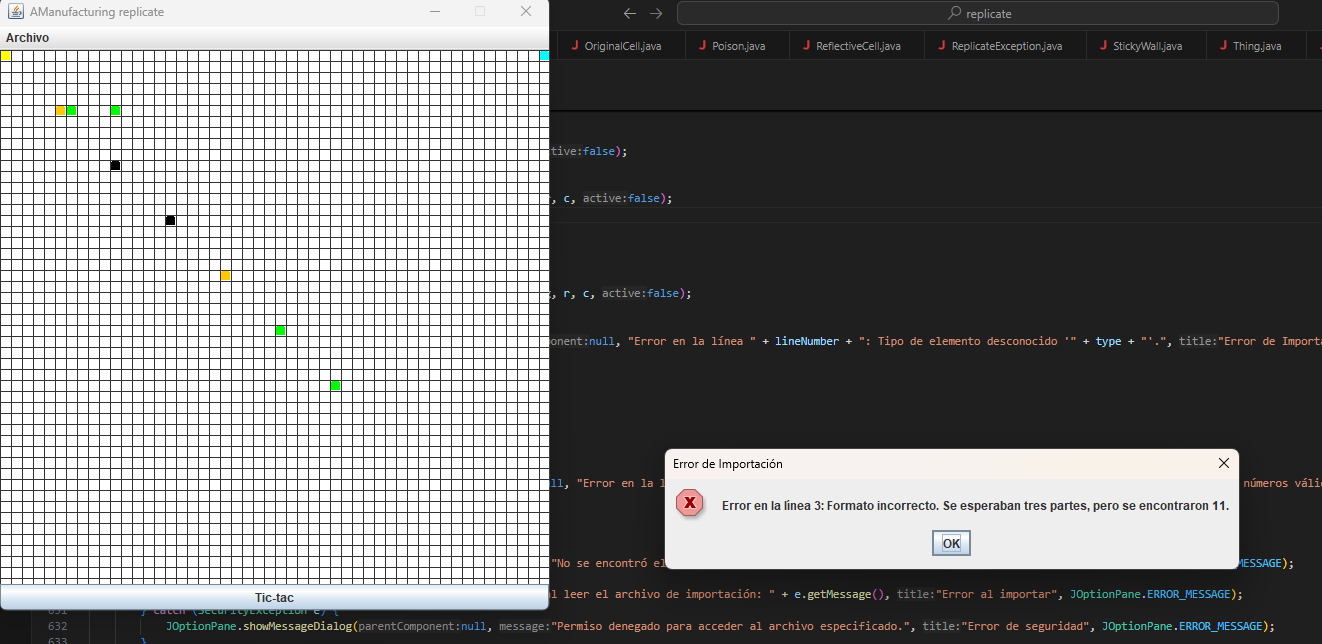
1. Escriban otro archivo con errores, llámelo replicateErr.txt, para ir arreglándolo con ayuda de su “importador”. Presente las pantallas que contengan los errores.

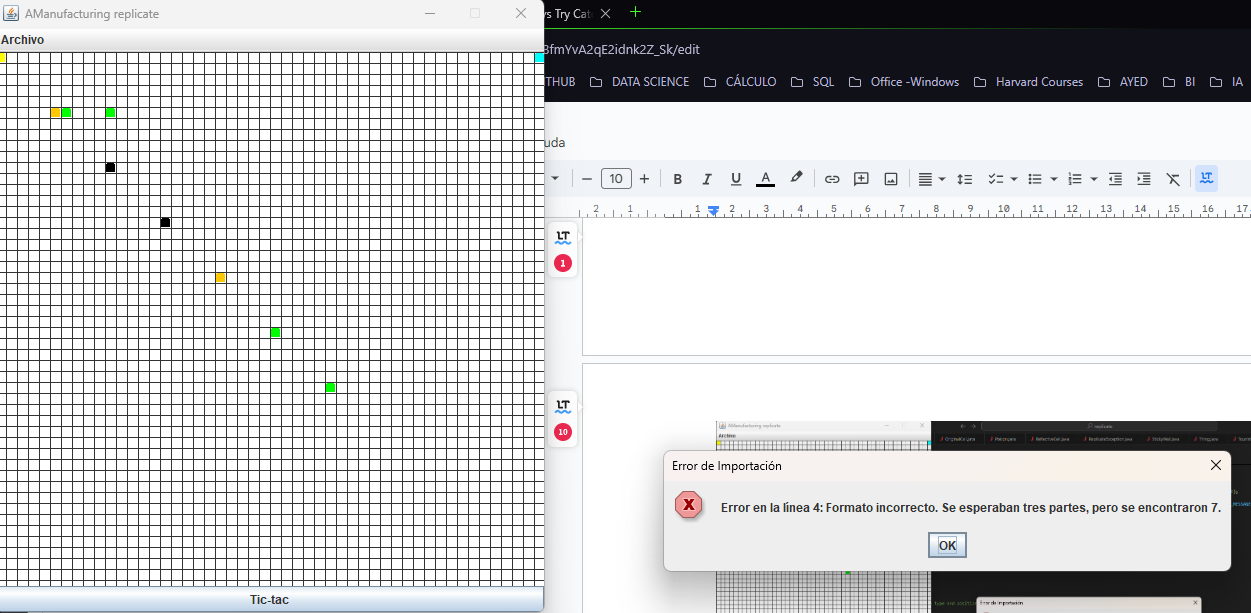
Ahora crear un archivo llamado replicateErr.txt con varios errores para que funcione como mini-compilador, a la hora de mostrar los errores.

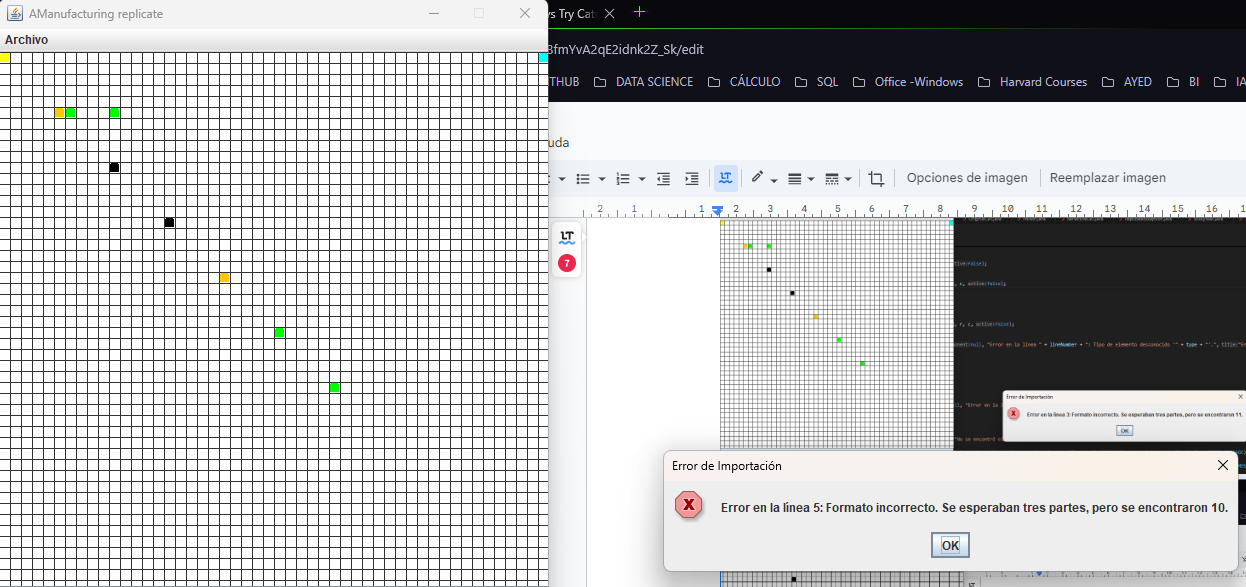


Y ahora, si se importa ese archivo, debería mostrar todos los errores.









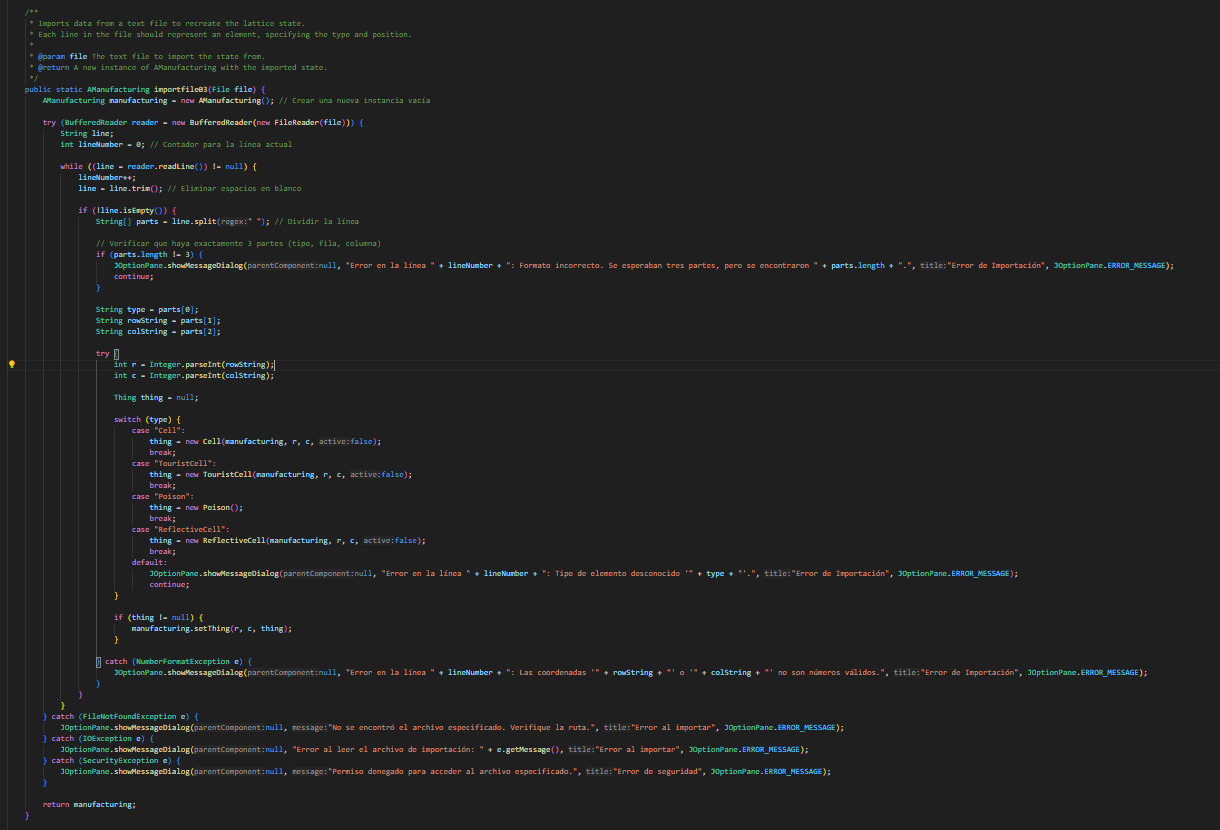
## BONO. Perfeccionando importar. Hacia un minicompilador flexible.

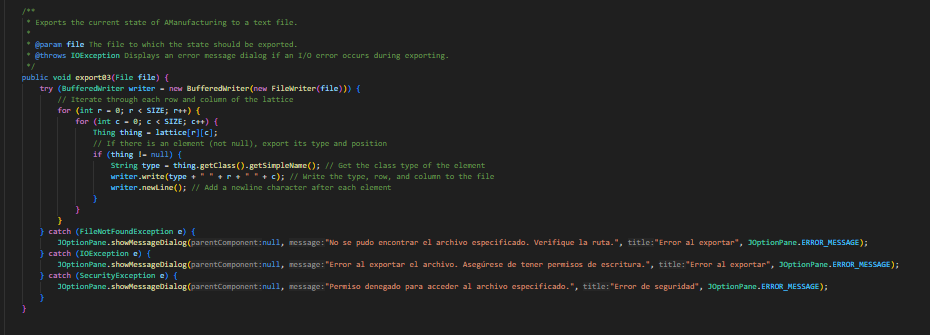
**[En lab06.doc, \*.asta , replicateErrG.txt \*.java] [NO OLVIDEN BDD y MDD]**

1. Copien las versiones actuales de import y export y renómbrenlos como import03 y

export03

Siguiendo con lo establecido:



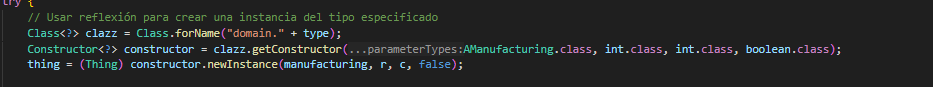


1. Perfeccionen los métodos import y export para que pueda servir para cualquier tipo de elementos creados en el futuro. No olviden pruebas de unidad.

(Investiguen cómo crear un objeto de una clase dado su nombre)

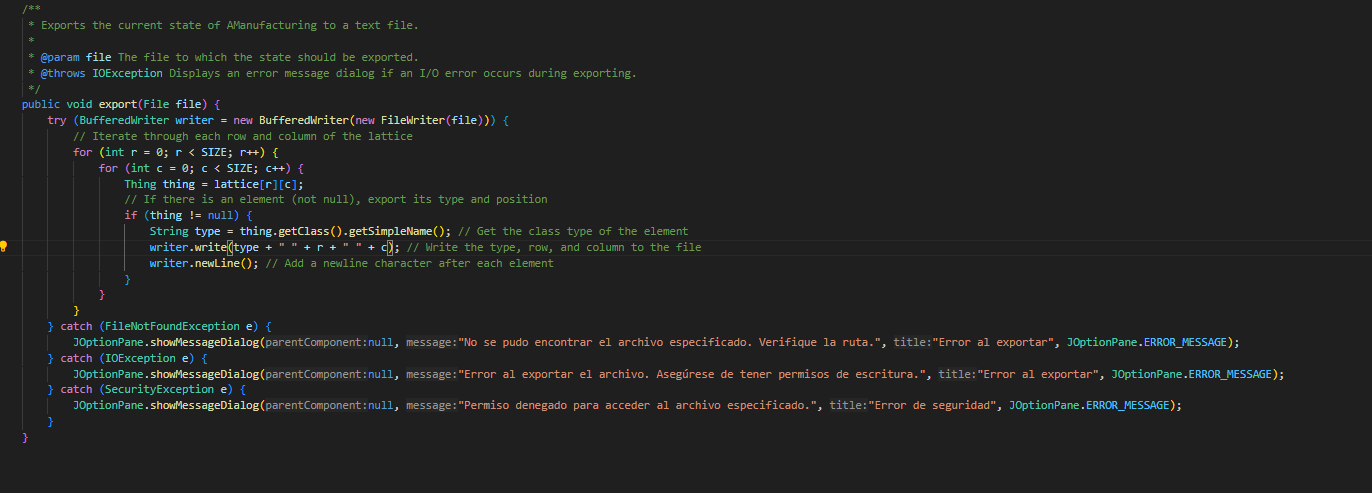
Partiendo de lo solicitado, se implementa los métodos. Al analizar se concluyó que el método ya es flexible, así como estás, ya que, ya está el archivo dentro de su clase, solo lo guarda con un formato en específico.

Para crear un objeto de una clase dado su nombre se requiere de:



Ahora, un pantallazo de los métodos a implementar:

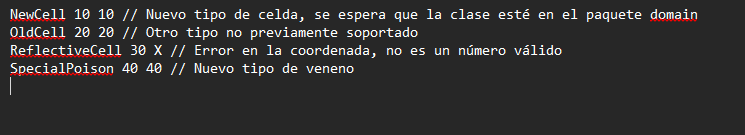




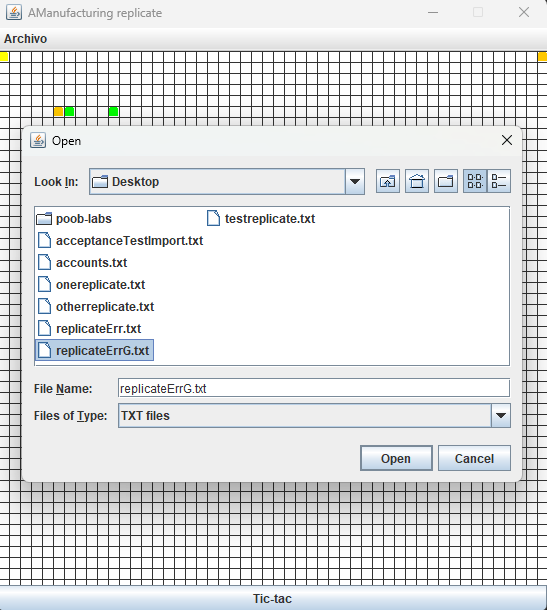
1. Escriban otro archivo de pruebas, llámelo replicateErrG.txt, para probar la flexibilidad. Presente las pantallas que contengan un error significativo.

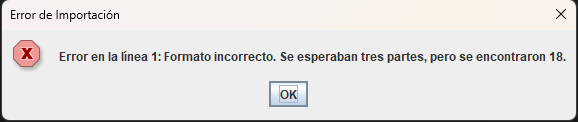
Listo, ahora para probar estos métodos aplicados al archivo a editar replicateErrG.txt se evidencia el correcto comportamiento de las excepciones para cualquier tipo de elementos creados en el futuro.

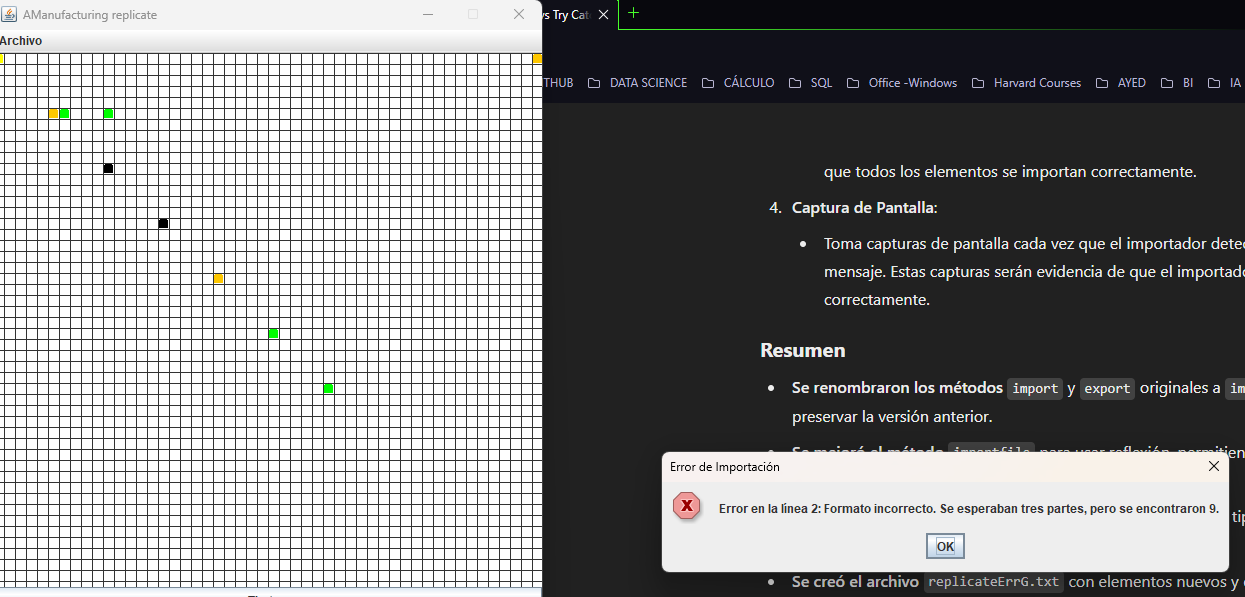
Con:

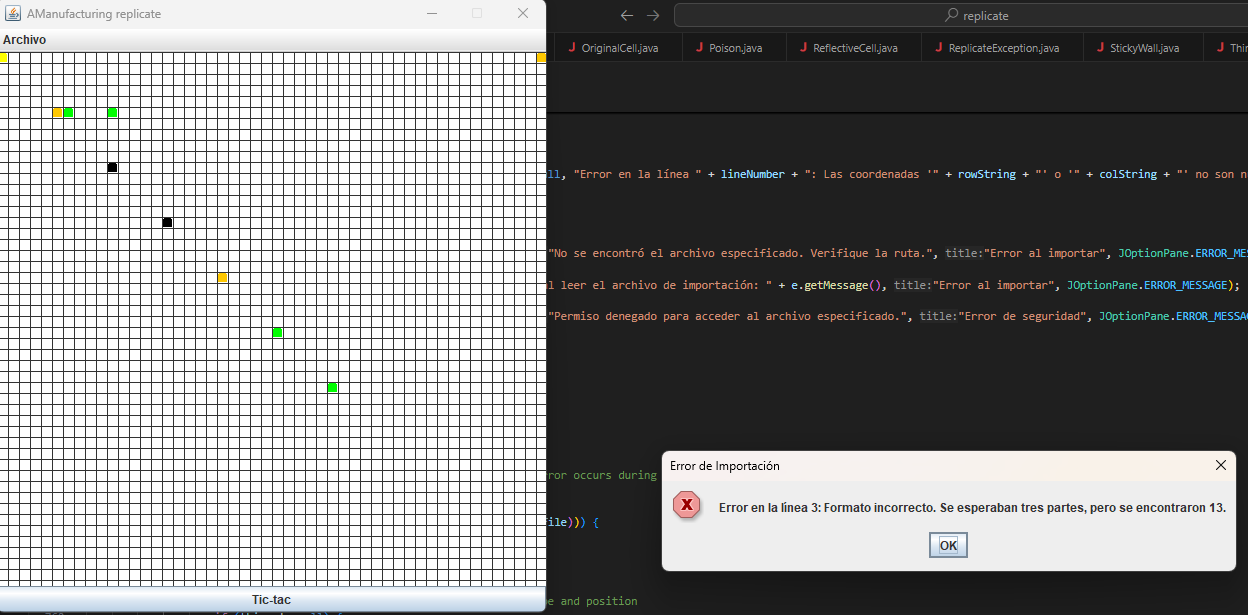


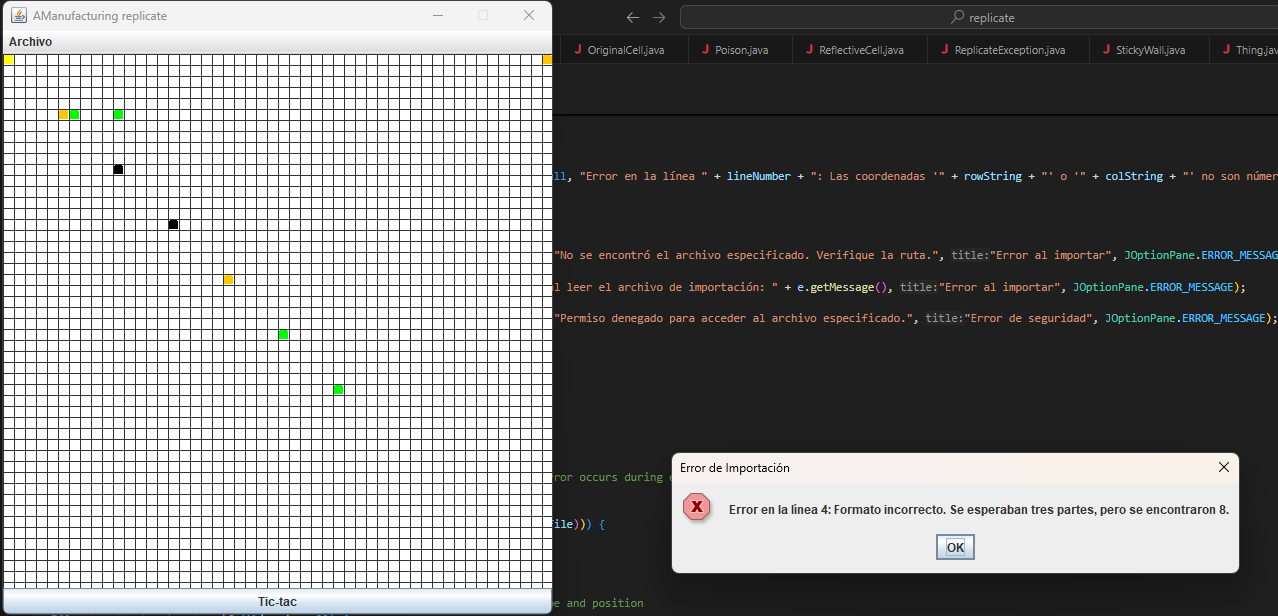
Se intenta importar este archivo para ver los errores que ocurren:





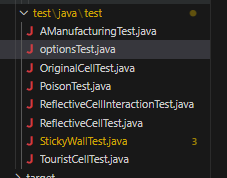


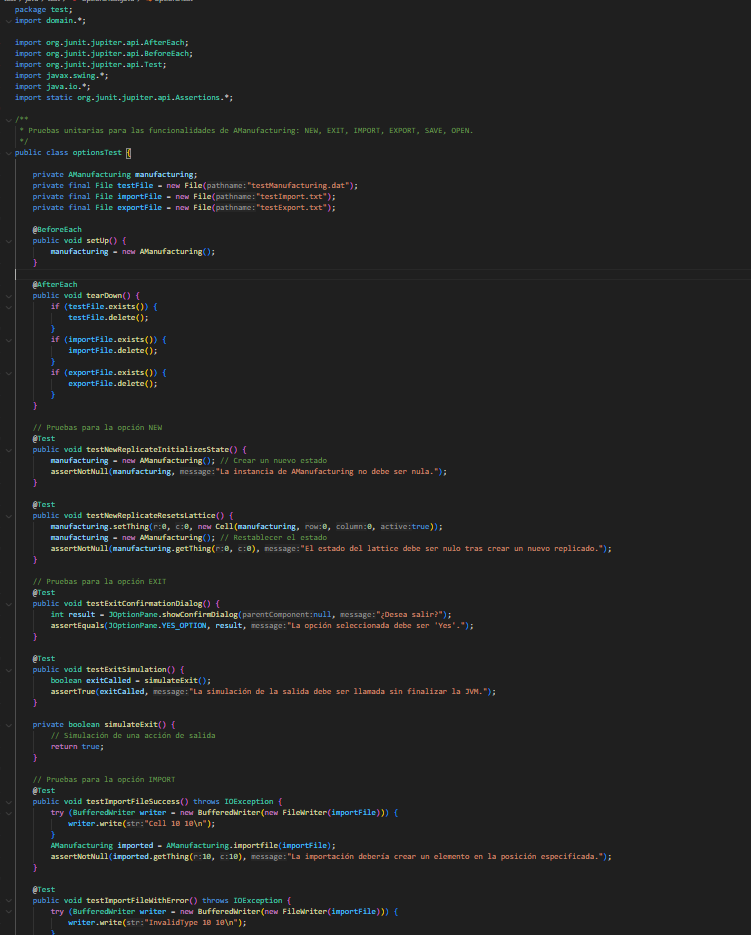




Vemos que todo funcionó correctamente, con su respectivo error en cada línea.

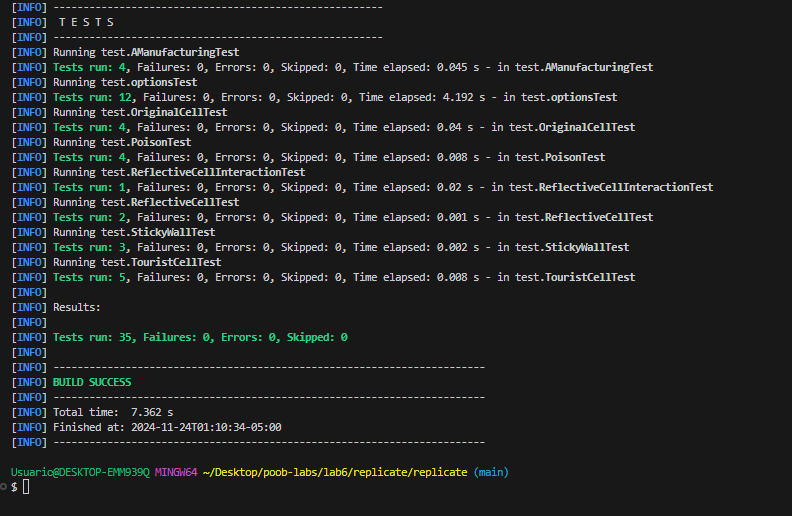
Luego, ahora se crean todas las pruebas unitarias para todas las opciones usadas en este laboratorio, OPEN, EXIT, SAVE, NEW, IMPORT, EXPORT



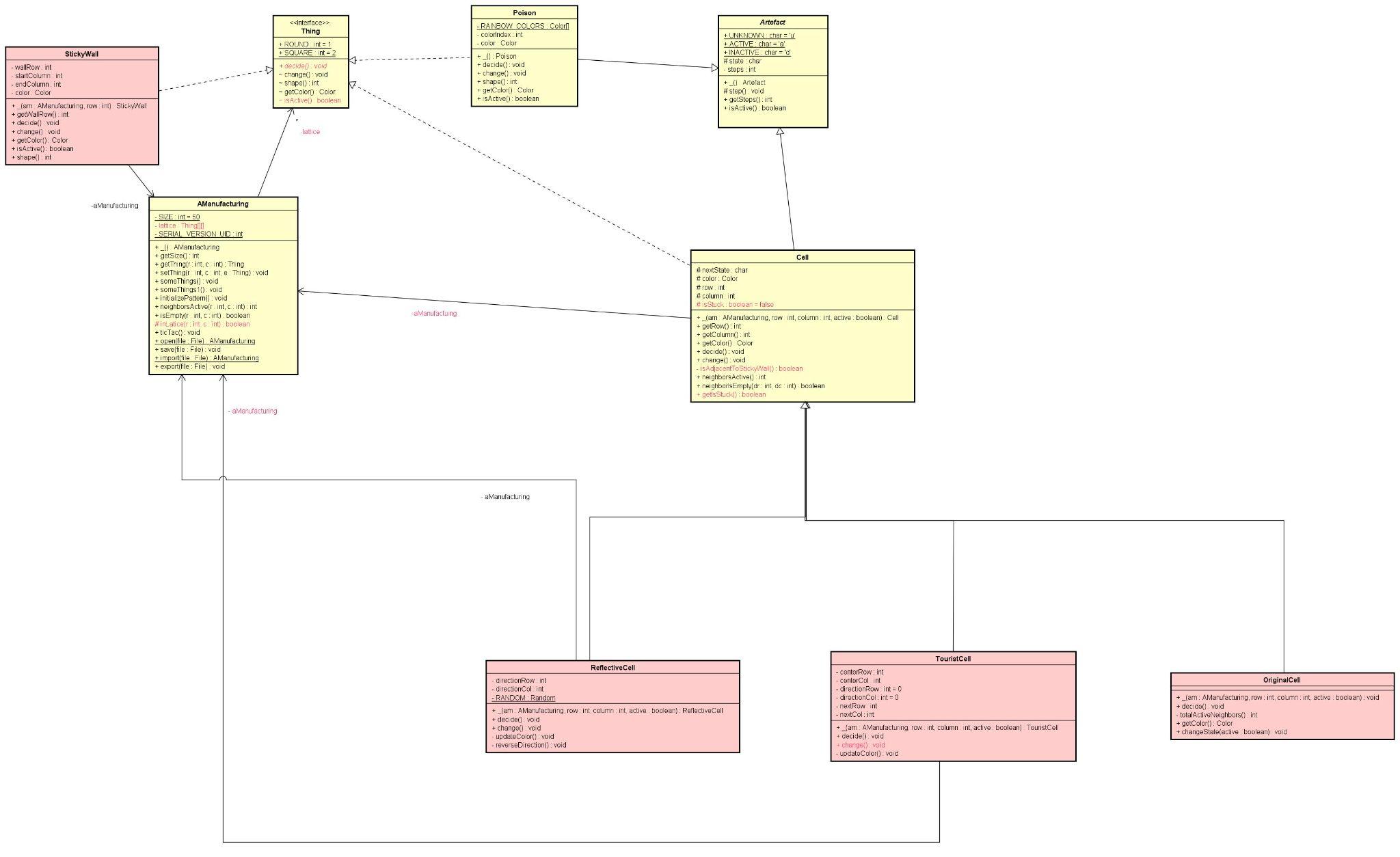


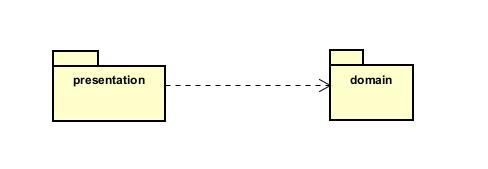


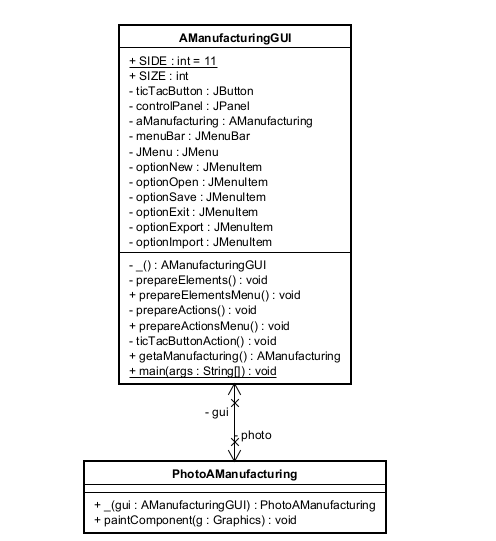
Se ve que todas las pruebas son exitosas, incluyendo las de las células



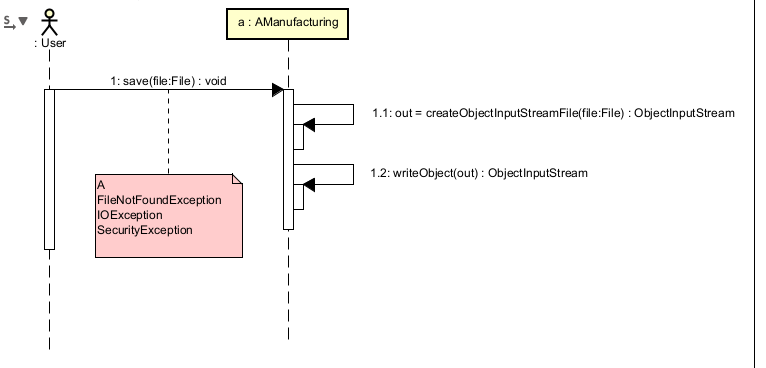
# Ahora, una vista del diseño (diagrama de clases y de secuencia y de paquetes).

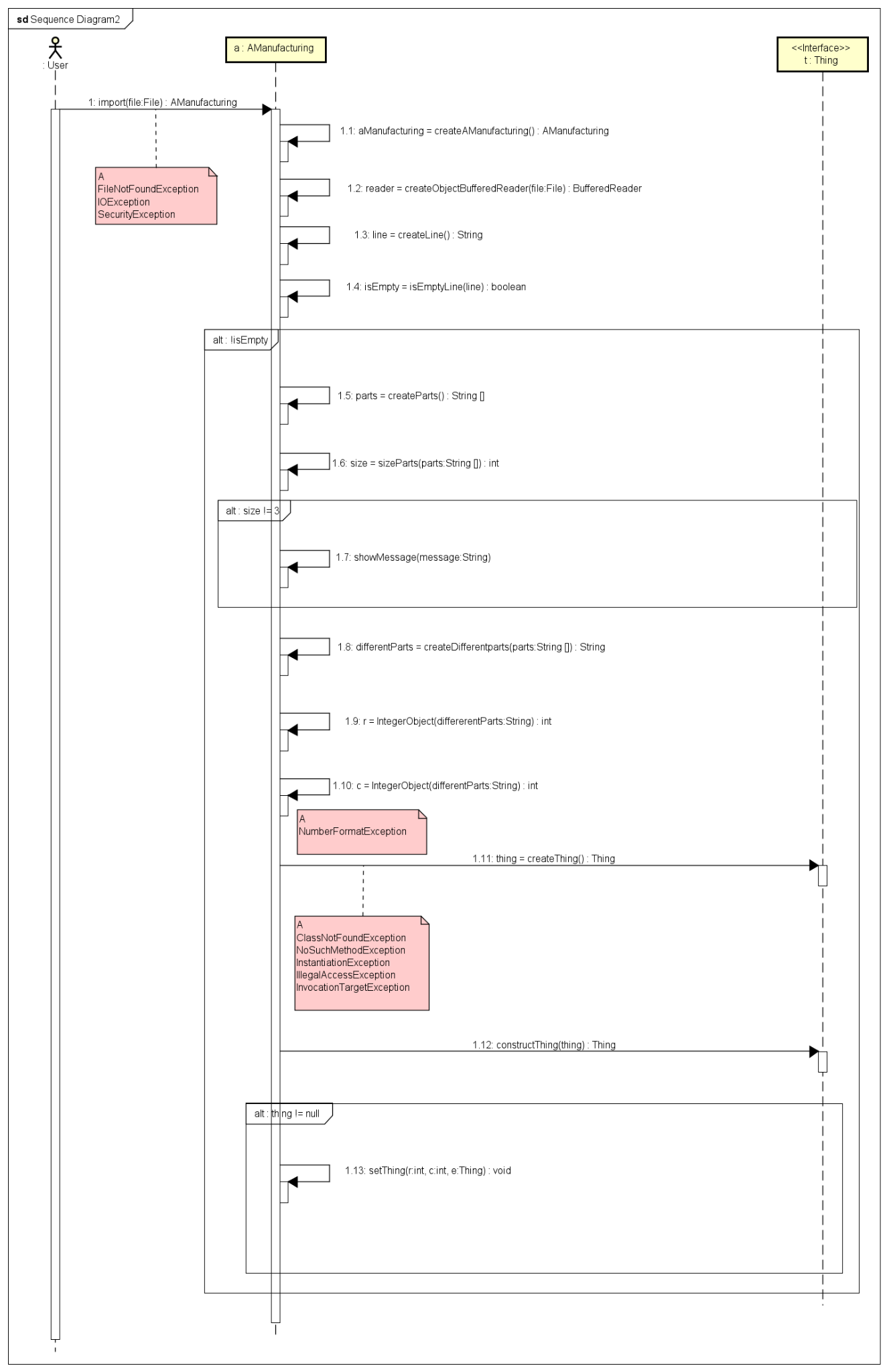


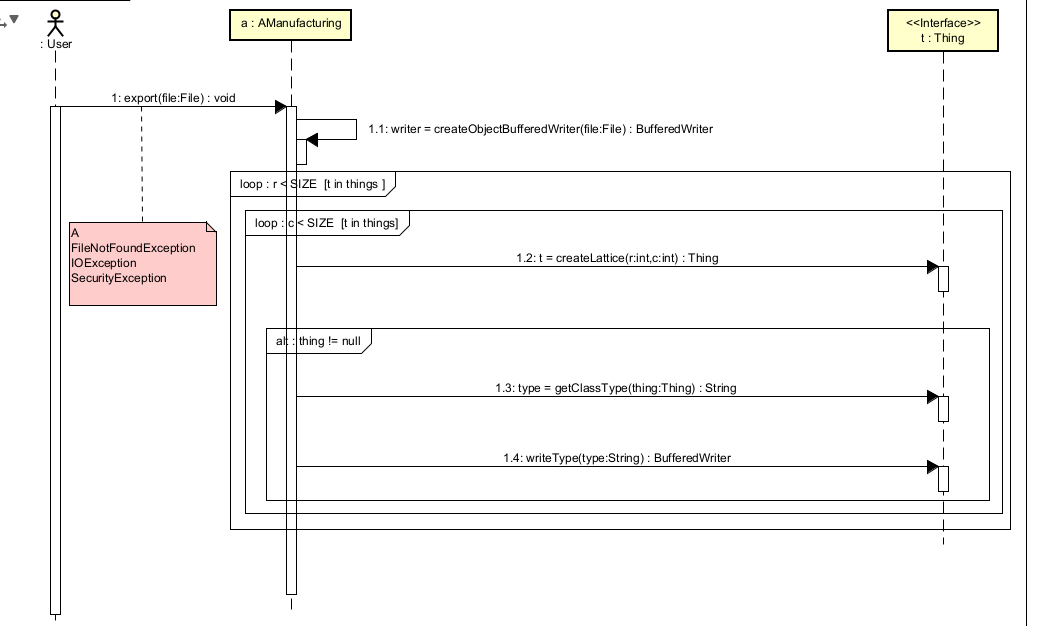




# 







# RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)

**El tiempo invertido fue de 20 horas.**

1. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

**Estado terminado porque mientras se avanzaba de punto, se iban cumpliendo los objetivos propuestos inicialmente.**

1. Considerando las prácticas XP del laboratorio. ¿Cuál fue la más útil? ¿Por qué?

**La práctica XP más útil en este lab fue Collective Ownership, ya que, aplicamos esta frase “The way this works is for each developer to create unit tests for their code as it is developed”. Cada vez que cada desarrollador hacía una parte del código (o algún funcionamiento en general), entonces se hacían pruebas unitarias para validar que el/los método/s funcionen para cualquier caso.**

1. ¿Cuál consideran fue el mayor logro? ¿Por qué?

**El mayor logro de este y todos los laboratorios fue terminarlos, porque la constante investigación y práctica hacia nuevas definiciones de POOB, hace que cada lab dedique más tiempo; y así mismo más conocimiento adquirido.**

1. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

**El mayor problema técnico fue entender bien lo de BufferedWriter y BufferedReader para poder lograr la última versión de import y export. Se logró investigando por distintos recursos.**

1. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

**Siempre el trabajo en equipo es vital para este y todos los labs; hubo constante comunicación usando Pair Programming. Y para mejorar resultados, se necesita de estar en constante práctica de todo lo aprendido, para que no se olvide.**