**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Persistencia**

**2023-02**

**Laboratorio 6/6 [ :) ]**

**Cristian Alvarez y Juliana Briceño**

# 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 : [Refactor](http://www.extremeprogramming.org/rules/refactor.html) whenever and wherever possible.



All code must pass all [unit tests](http://www.extremeprogramming.org/rules/unittests.html) before it can be released.

# 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 colony adicionando un menú barra con las optiones básicas de entrada-salida y las optiones 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.

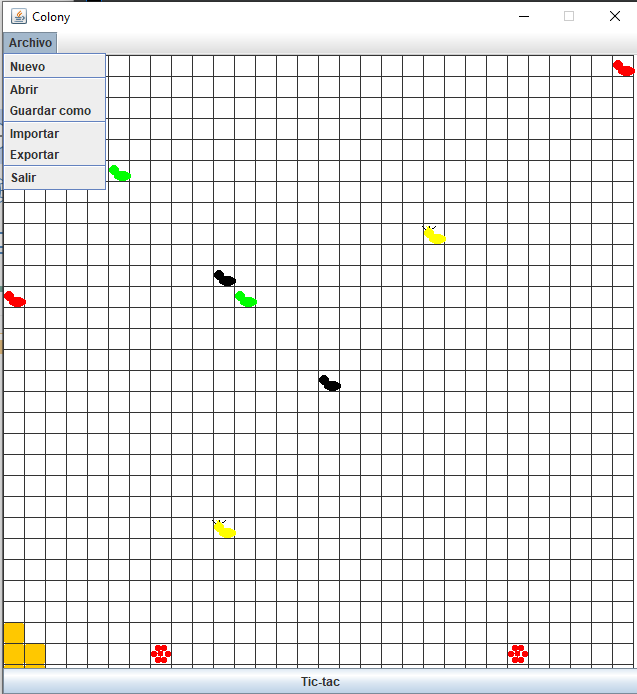
## Creando la maqueta

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

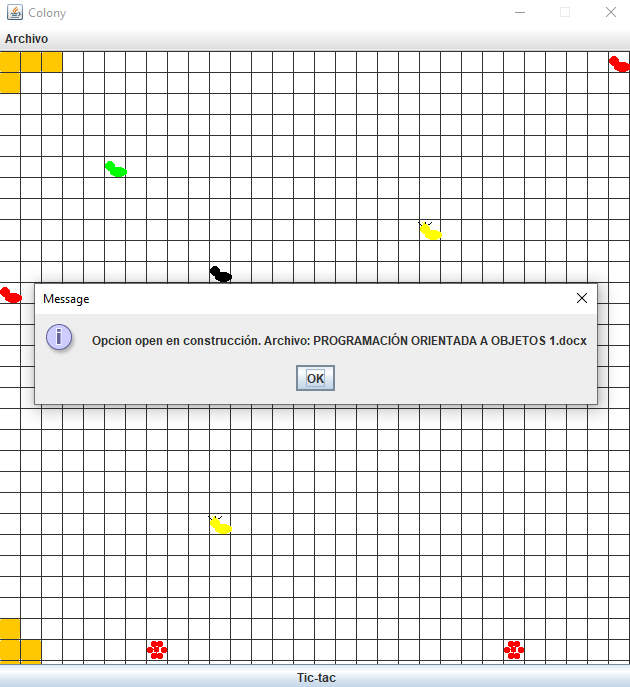
En este punto vamos a construir la maqueta correspondiente a esta extensión siguiendo el patrón MVC.

1. **MODELO:** Preparen en la clase fachada del modelo los métodos correspondientes a las cuatro optiones básicas de entrada-salida (open, save, import y export). Los métodos deben simplemente propagar una colonyException con el mensaje de “Opción nombreOpción en construcción. Archivo nombreArchivo”. Los métodos deben tener un parámetro File.
2. **CONTROLADOR:** Construyan los oyentes orrespondientes a las seis optiones. Para esto creen el método prepareAccionesMenu y los métodos base del controlador (optionOpen, optionSave, optionImport, optionExport, optionNew, optionExit), En las optiones que lo requieran usen un FileChooser y atiendan la excepción. Estos métodos llaman el método correspondiente de la capa de dominio.
3. **VISTA :** Construyan un menú barra que ofrezca, además de las optiones básicas de entrada-salida, las optiones estándar de nuevo y salir (Nuevo, Abrir, Guardar como, Importar, Exportar como, Salir). No olviden incluir los separadores. Para esto creen el método prepareElementsMenu. Ejecuten las diferentes acciones del menú y para cada una de ellas capture una pantalla significativa.

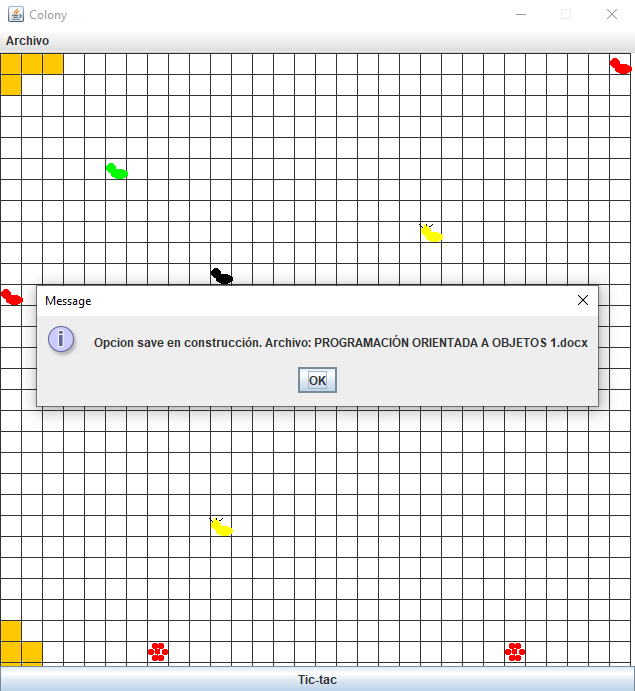
**MENU GENERAL**



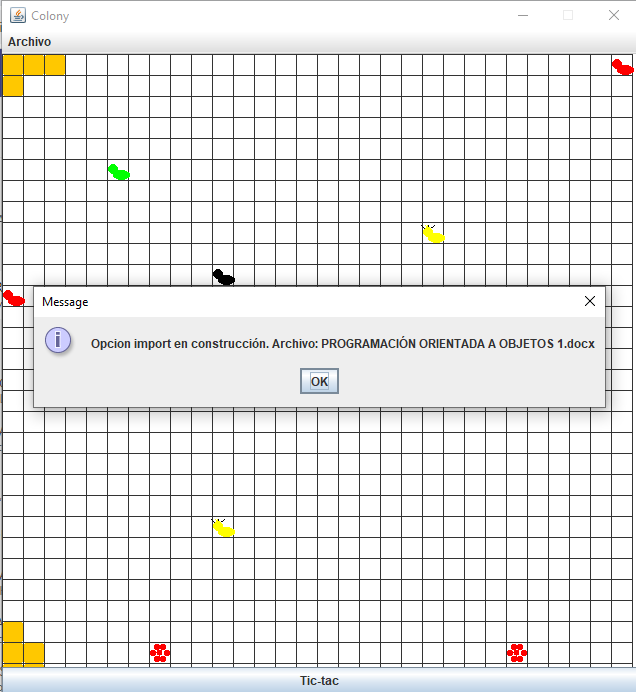
PROBANDO OPCION OPEN



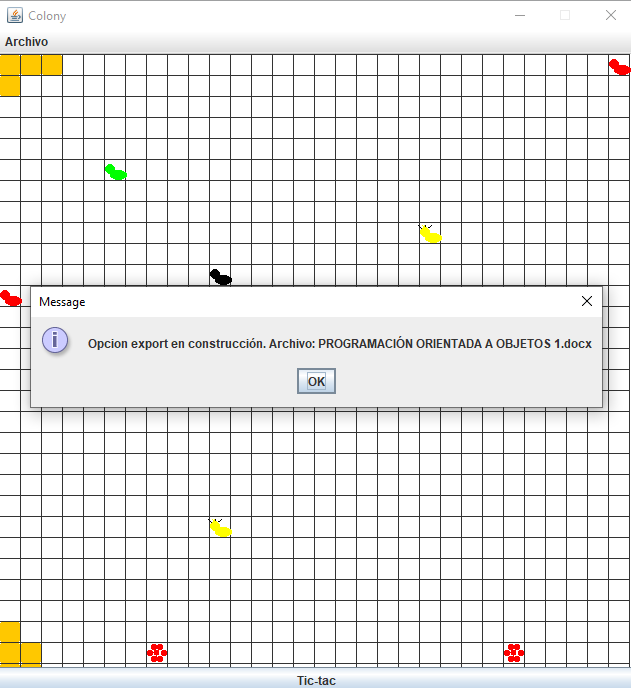
PROBANDO OPCION SAVE



PROBANDO OPCION IMPORT



PROBANDO OPCION EXPORT



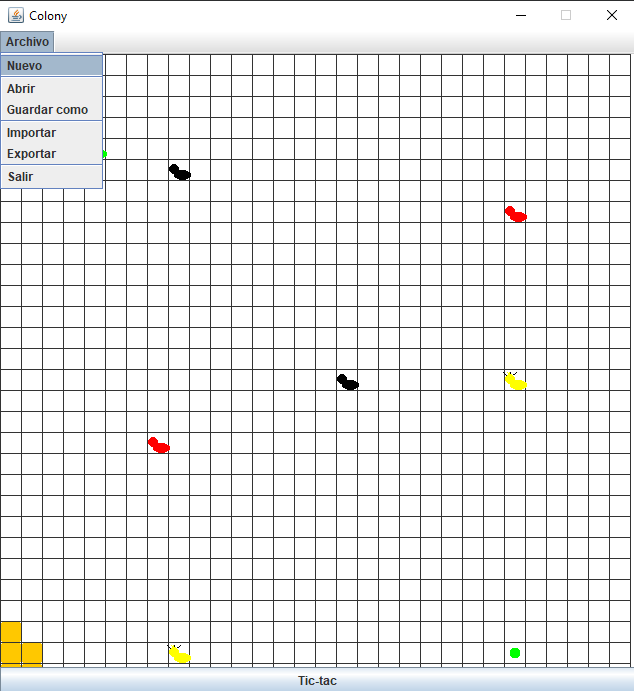
## Implementando salir y nuevo

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

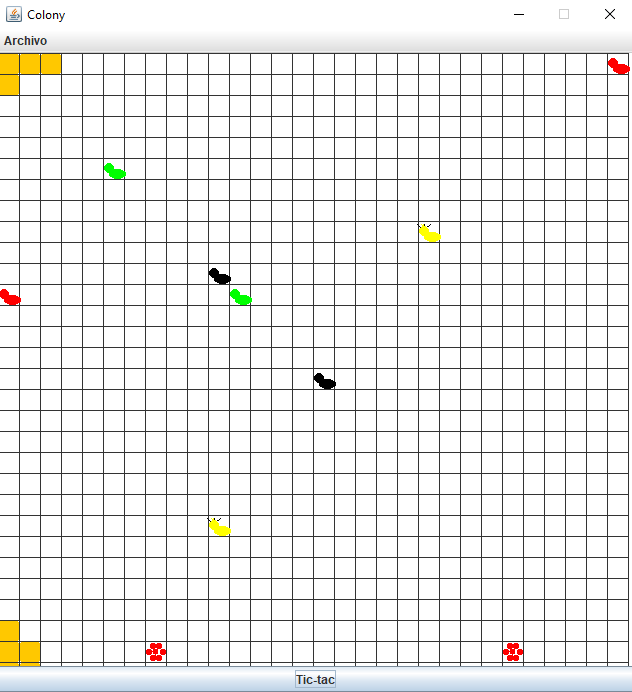
Las optiones salir e iniciar van a ofrecer los dos servicios estándar de las aplicaciones. El primero no requiere ir a capa de dominio y el segundo sí.

1. Construyan el método optionExit que hace que se termine la aplicación. No es necesario incluir confirmación.
2. Construyan el método optionNew que crea un nuevo colony. Capturen una pantalla significativa.

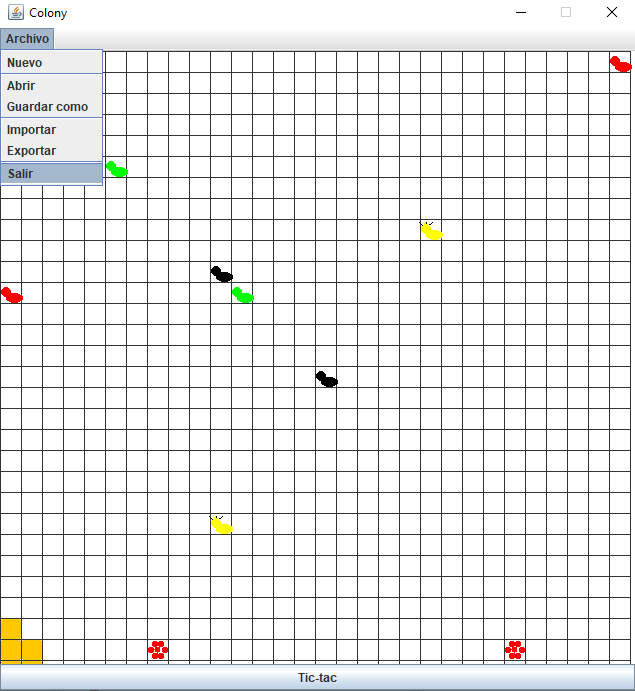
PROBANDO NEW



Verificando que se resetee el tablero



PROBANDO EXIT



## Implementando salvar y abrir

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

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

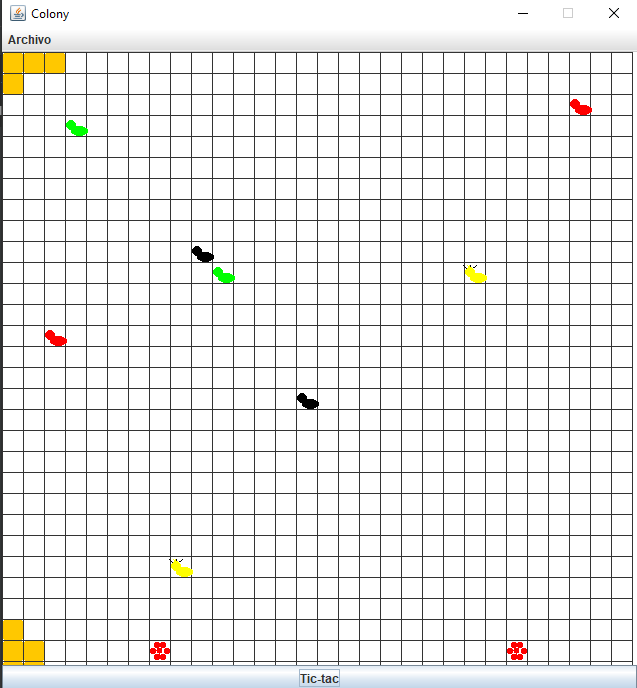
1. Copien las versiones actuales de open y save y renómbrenlos como open00 y save00
2. Construyan el método save que ofrece el servicio de guardar en un archivo el estado actual de colony. Por ahora para las excepciones sólo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.
3. Validen este método guardando el estado obtenido después de dos clics como oneColony.dat. ¿El archivo se creó en el disco? ¿Cuánto espacio ocupa?

Si se creo en el disco, y ocupa 3KB

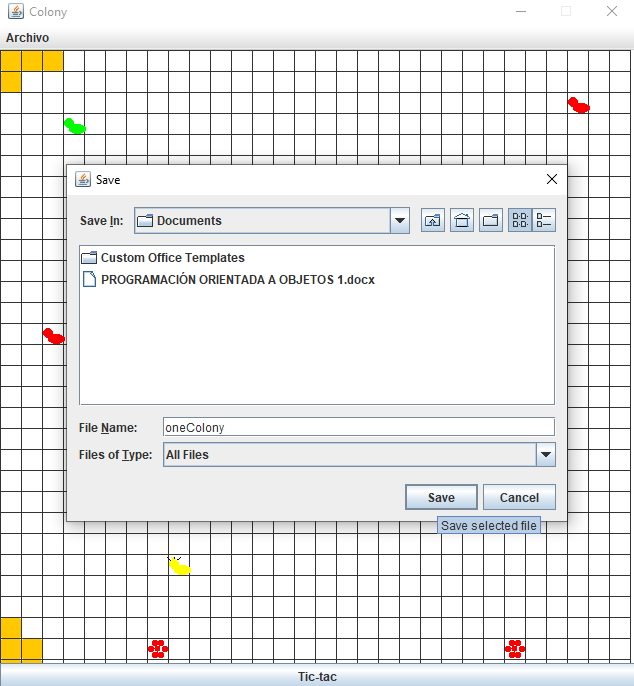


1. Construyan el método open que ofrece el servicio de leer un colony de un archivo. Por ahora para las excepciones sólo consideren un mensaje de error general. No olviden diseño y pruebas de unidad agregar Flower al astah.
2. Realicen una prueba de aceptación para este método iniciando la aplicación, creando un nuevo situación en el colony y abriendo el archivo oneColony.dat. Capturen imágenes significativas de estos resultados.

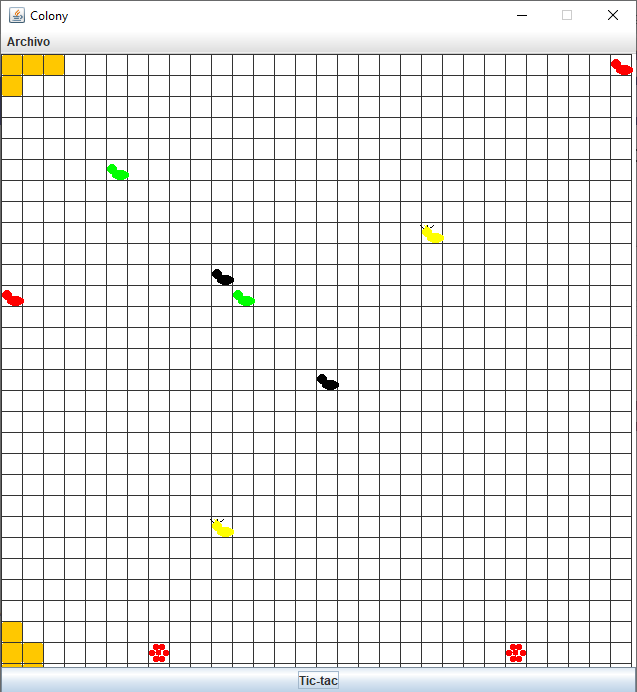
Estado inicial del oneColony después de hacer tic tac



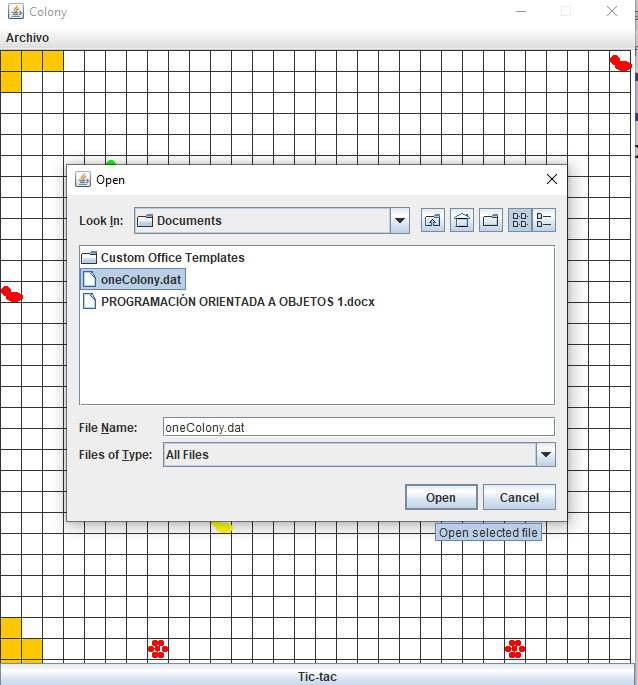
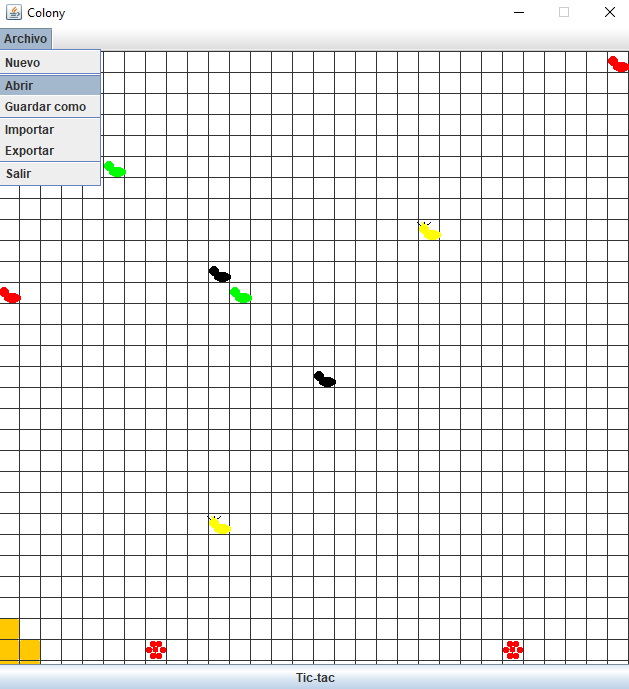
Guardando



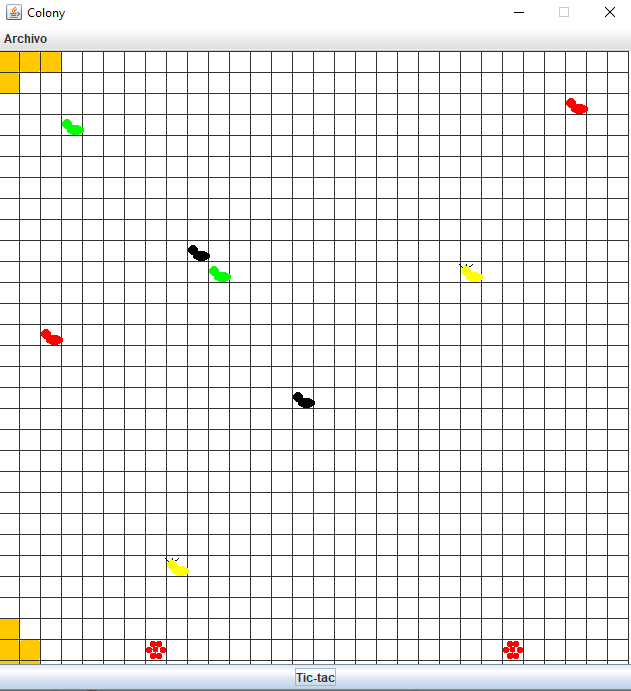
Nuevo colony



Abriendo



Abierto correctamente



## Implementando importar y exportar

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

Estas operaciones nos van a permitir importar información del colony 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

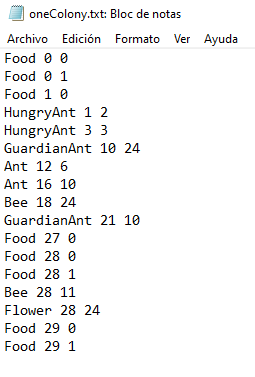
En cada línea asociada un elemento se especifica el tipo y la posición.

Ant 100 100

Flower 500 500

1. Copien las versiones actuales de import y export y renómbrenlos como import00 y export00
2. 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 sólo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.
3. Realicen una prueba de aceptación de este método: iniciando la aplicación y exportando como oneColony.txt. Editen el archivo y analicen los resultados. ¿Qué pasó?

R// Si editamos el archivo, cambian los valores y nos pide guardar los cambios

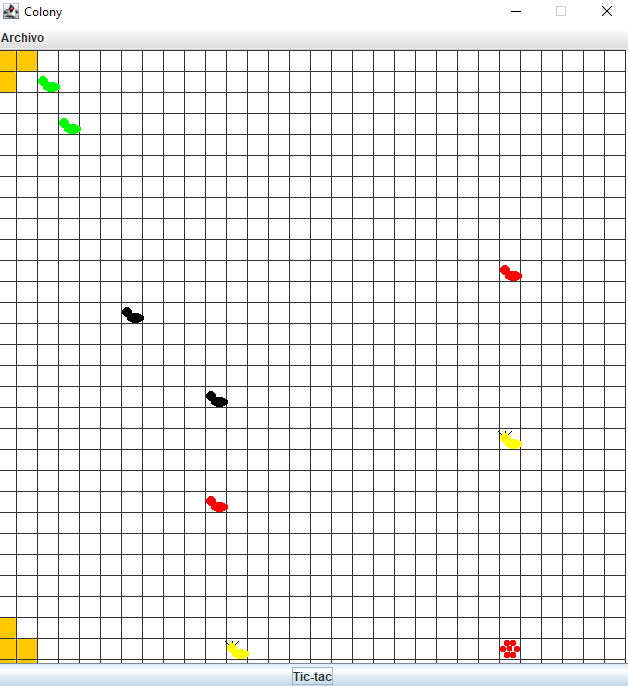


1. Construyan el método import que ofrece el servicio de importar de un archivo texto con el formato definido. Por ahora sólo considere un mensaje de error general. No olviden diseño y pruebas de unidad.

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

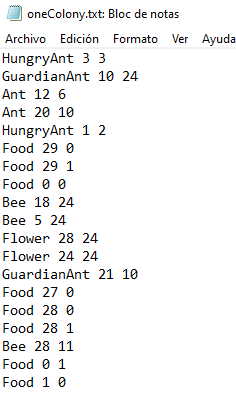
1. Realicen una prueba de aceptación de este par de métodos: iniciando la aplicación exportando a oneColony.txt. saliendo, entrando, creando una nueva e importando el archivo otherColony.txt. ¿Qué resultado obtuvieron? Capturen la pantalla final.

Obtuvimos el estado esperado

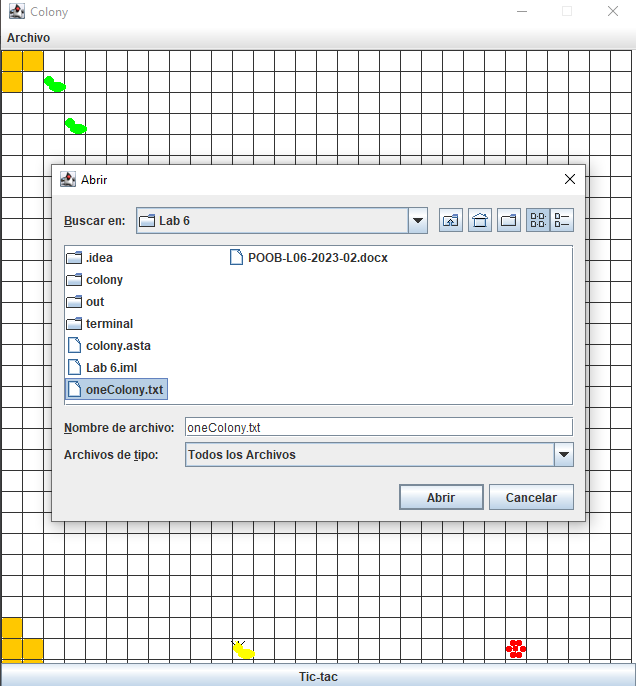


1. Realicen otra prueba de aceptación de este método escribiendo un archivo de texto correcto en oneColony.txt. e import este archivo. ¿Qué resultado obtuvieron? Capturen la pantalla.

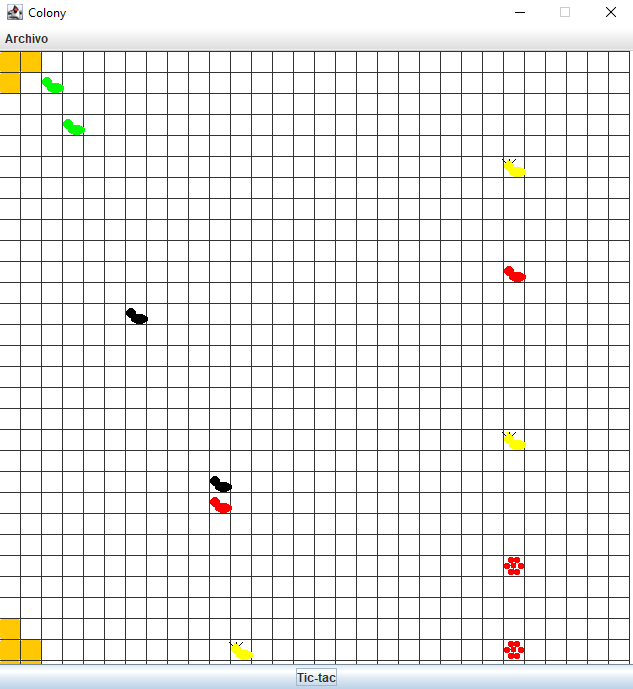
Escribimos un txt de manera correcta



Ahora importamos este archivo



Es el resultado esperado



## 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

R//: Se guarda el objeto en sí y cuando se abre el .dat se utiliza ese objeto guardado para que se pinte correctamente el estado del tablero.

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

R//: Se guardan líneas de texto con la información relevante del tablero, luego cuando se abre el .txt leemos las líneas de texto para crear los correspondientes objetos de la aplicación.

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

comportamiento 1: Objetos

comportamiento 2: Texto plano

**Tipo de Datos Almacenados:**

* Objetos: ObjectInputStream y ObjectOutputStream permiten almacenar y recuperar objetos Java completos. Puedes escribir y leer cualquier objeto serializable, incluyendo objetos personalizados que implementen la interfaz Serializable.
* Texto: FileReader y FileWriter se utilizan para leer y escribir caracteres en formato de texto. Generalmente, se trabaja con datos en formato de cadena (String) o caracteres, y no se pueden almacenar directamente objetos en formato binario.

**Formato de Almacenamiento:**

* Objetos: Los archivos de objetos almacenan datos en formato binario, lo que significa que los datos se guardan en un formato que solo entiende la máquina.
* Texto: Los archivos de texto almacenan datos en formato de texto legible por humanos. Cada valor se representa como una cadena de caracteres y se separa por delimitadores (como espacios, comas o tabulaciones).

**Serialización y Deserialización:**

* Objetos: Antes de almacenar objetos en un archivo de objetos, estos deben ser serializados utilizando la interfaz Serializable. La serialización convierte el objeto en una secuencia de bytes que luego puede ser almacenada y posteriormente deserializada para recrear el objeto.
* Texto: No hay un proceso de serialización/deserialización directo en archivos de texto. Los datos se escriben y leen como caracteres, y cualquier transformación de datos necesaria debe ser manejada manualmente.

**Uso de Excepciones:**

* Objetos: Al trabajar con ObjectInputStream y ObjectOutputStream, generalmente manejarás excepciones como ClassNotFoundException y IOException.
* Texto: Al usar FileReader y FileWriter, es común manejar excepciones como FileNotFoundException y IOException.

**Facilidad de Lectura (humano):**

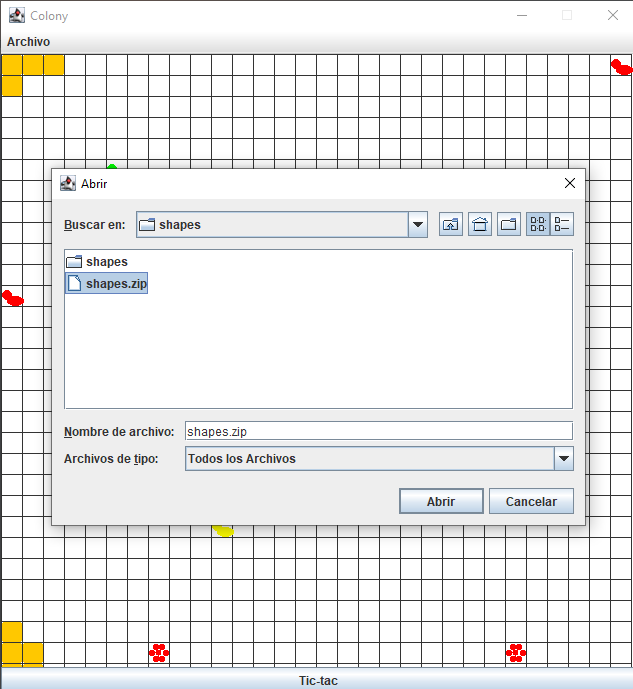
* Objetos: Los archivos de objetos son binarios y no son legibles para los humanos. Son más eficientes para almacenar y recuperar datos, pero no son adecuados para la edición manual del archivo.
* Texto: Los archivos de texto son legibles para los humanos y se pueden editar fácilmente con un editor de texto. Sin embargo, pueden ser menos eficientes en términos de espacio de almacenamiento y tiempo de lectura/escritura.

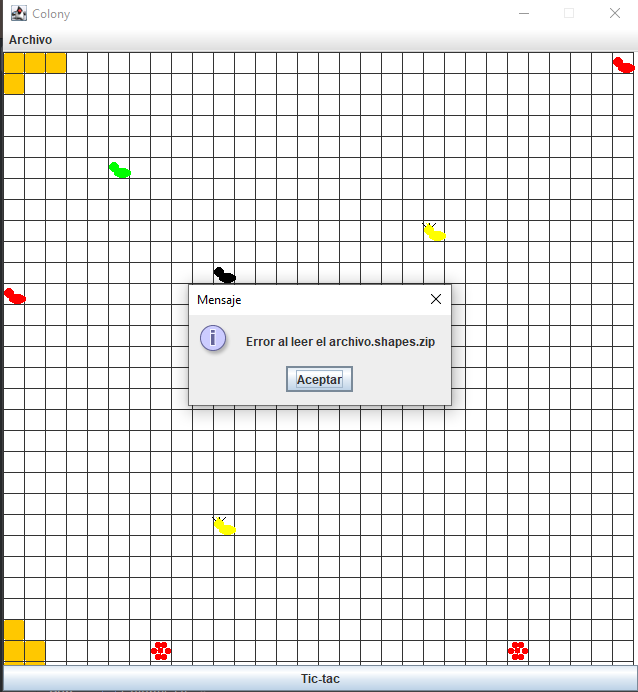
## Perfeccionando salvar y abrir

### [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
2. Perfeccionen el manejo de excepciones de los métodos open y save detallando los errores.
3. Realicen una prueba de aceptación para validar uno de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.

Probando el Error al leer el archivo.

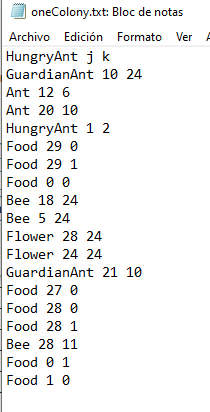


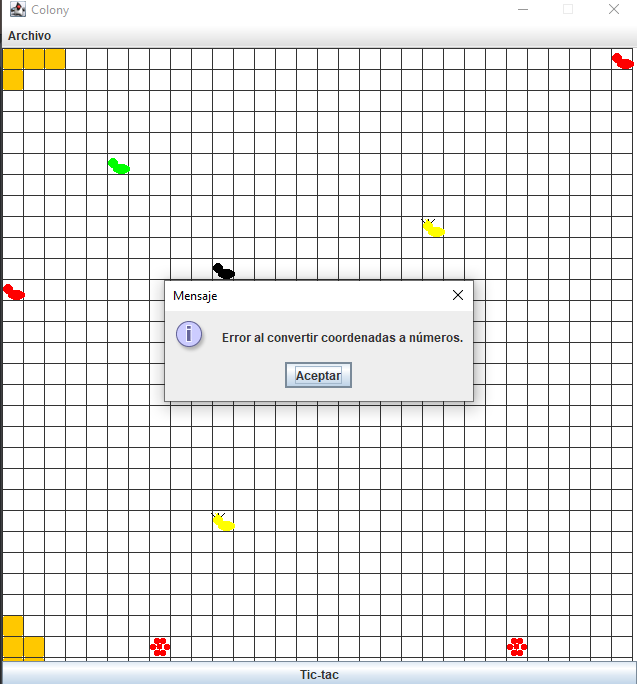


**Perfeccionando importar y exportar.**

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

1. Copien las versiones actuales de import y export y renómbrenlos como import01 y export01
2. Perfeccionen el manejo de excepciones de los métodos import y export detallando los errores.
3. Realicen una prueba de aceptación para validar uno de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.

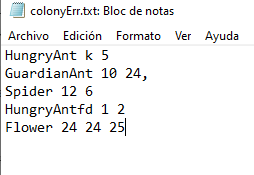


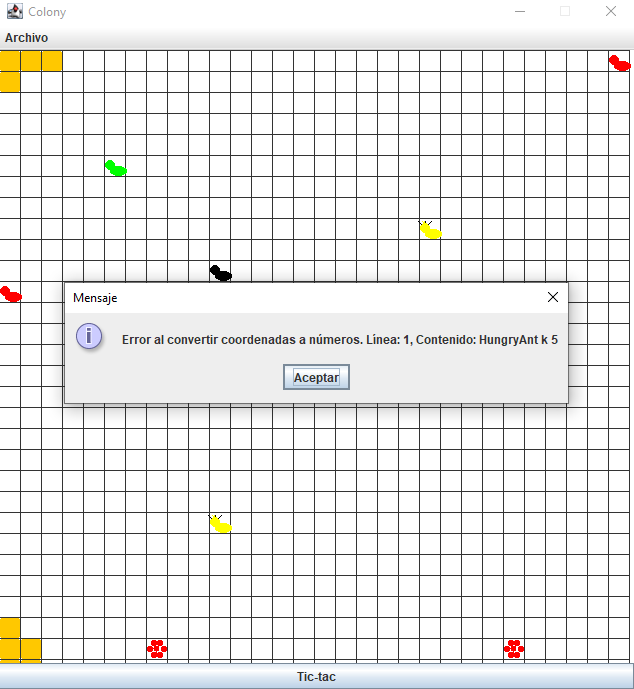


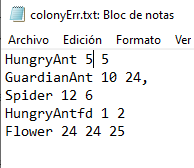
**Perfeccionando importar. Hacia un minicompilador.**

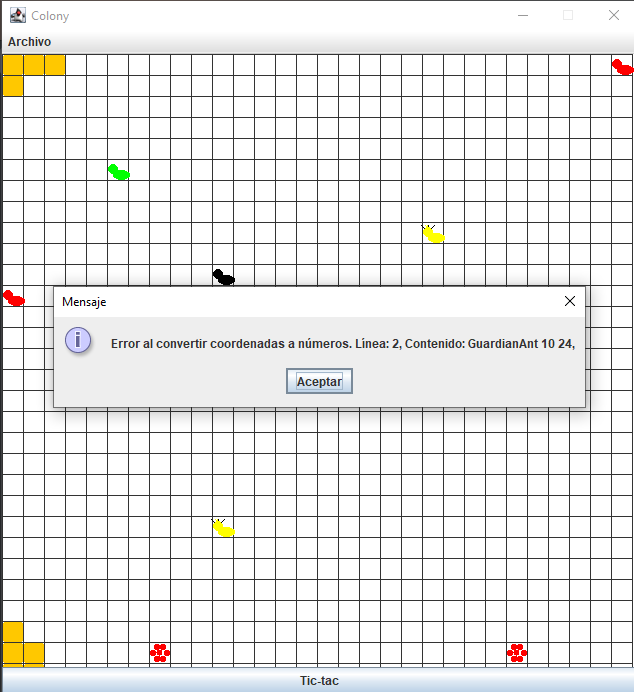
### [En lab06.doc, \*.asta , colonyErr.txt \*.java] [NO OLVIDEN BDD y MDD]

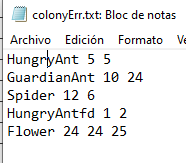
1. Copien las versiones actuales de import y export y renómbrenlos como import02 y export02
2. Perfeccionen el método **import** para que, además de los errores generales, en las excepciones indique el detalle de los errores encontrados en el archivo (como un compilador) : número de línea donde se encontró el error, palopen que tiene el error y causa de error.
3. Escriban otro archivo con errores, llámelo colonyErr.txt, para ir arreglándolo con ayuda de su “importador”. Presente las pantallas que contengan los errores.

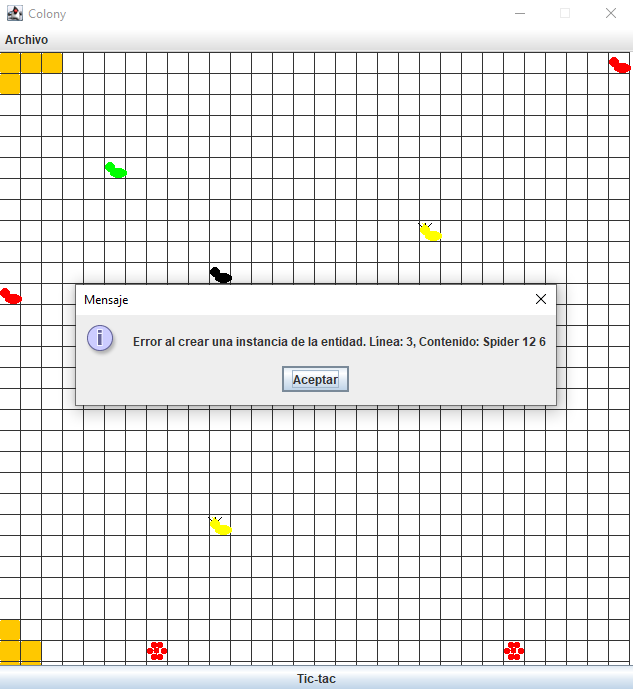


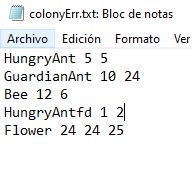


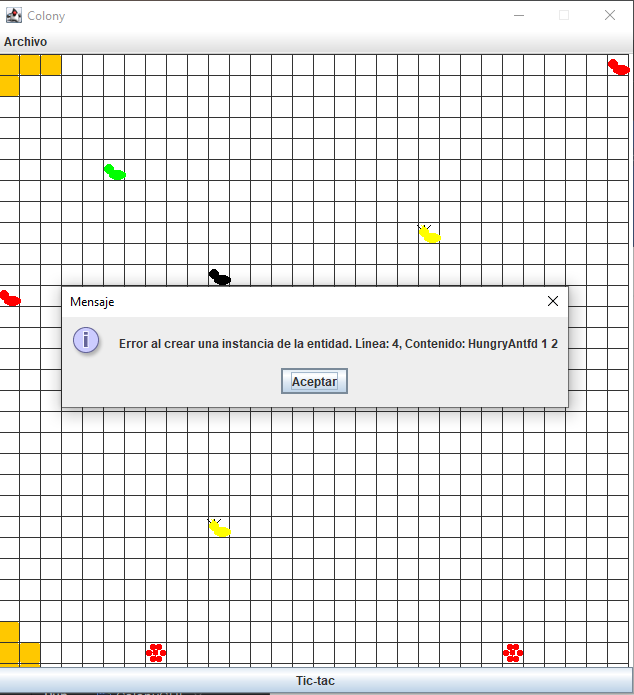


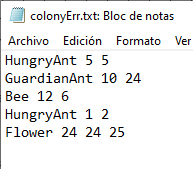


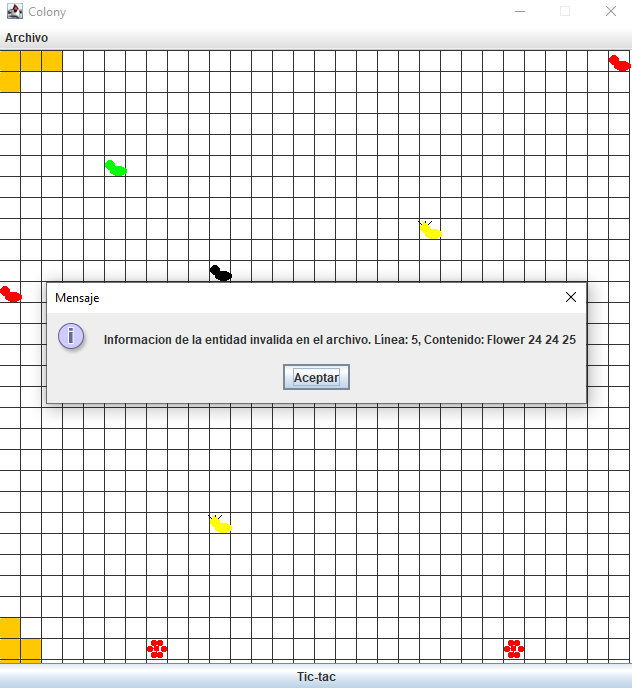


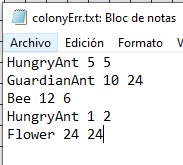


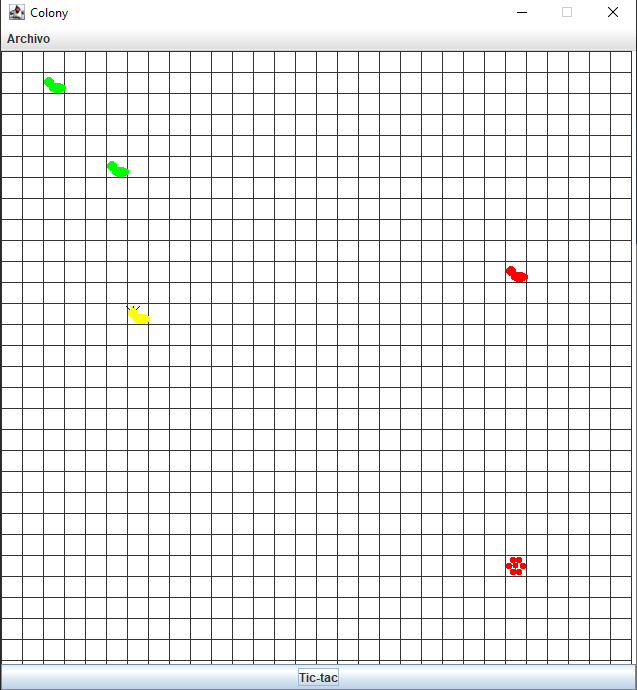










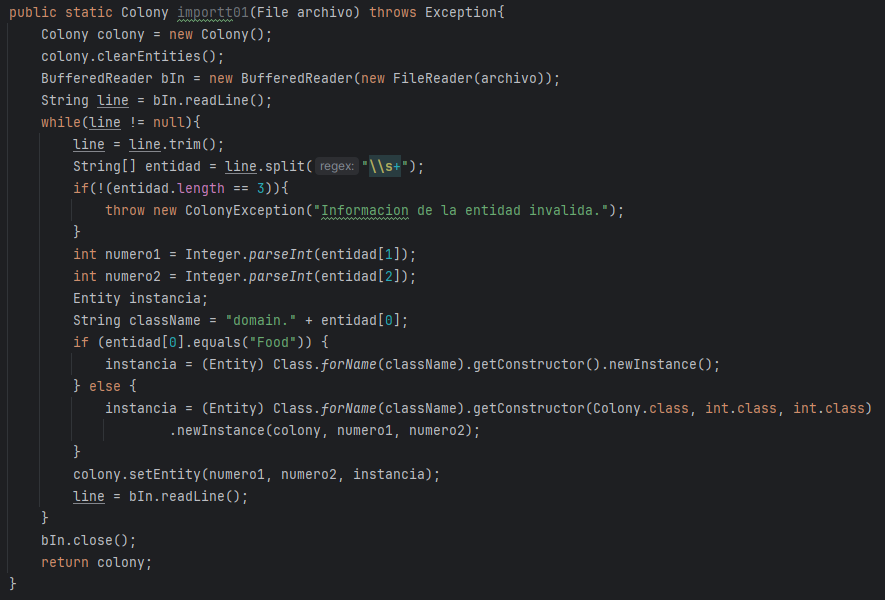


**BONO. Perfeccionando importar. Hacia un minicompilador flexible.**

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

1. Copien las versiones actuales de import y export y renómbrenlos como import03 y export03
2. Perfeccionen los métodos import y export para que pueda servir para cualquier tipo de elementoscreados en el futuro

R// Desde la segunda versión de import, pensamos en una versión extensible, asi que ya esta implementada



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

Nuestro código es muy extensible por diversas razones:  
****Reflexión y Creación Dinámica de Objetos:****

**En el método *importt*, se utiliza reflexión para crear dinámicamente instancias de clases basándose en el nombre de la clase proporcionado en el archivo de entrada. Esto significa que si se añade una nueva clase en el paquete domain, el código existente no necesita ser modificado. La nueva clase será automáticamente detectada y utilizada, siempre y cuando cumpla con las expectativas de tener un constructor sin parámetros o un constructor que tome una instancia de Colony y dos enteros.**

****Manejo de Excepciones:****

**El manejo de excepciones personalizado (ColonyException) proporciona una manera clara de gestionar problemas específicos relacionados con la lógica de la colonia. Si se agregan nuevas entidades y surge algún problema específico al importar o exportar esas entidades, simplemente se lanza una ColonyException con un mensaje descriptivo sin afectar la estructura del código.**

****No Dependencia en Implementaciones Específicas:****

**Los métodos *importt* y *export* no dependen de implementaciones específicas de las clases de entidad. Utilizan la interfaz Entity y la reflexión para trabajar con clases de entidad de manera genérica. Esto facilita la adición de nuevas entidades sin modificar directamente estos métodos.**

****Abstracción del Formato de Entrada/Salida:****

**La lectura y escritura de datos en el método *importt* y *export* se realiza de una manera generalizada que no está acoplada a un formato específico. Esto permite que estos métodos sean utilizados con diferentes formatos de entrada/salida sin necesidad de cambios extensos.**

****Uso de Interfaces y Abstracción:****

**La interfaz Entity proporciona una abstracción que facilita la adición de nuevas entidades. Al programar contra la interfaz, en lugar de las implementaciones concretas, se facilita la extensibilidad y el cambio de implementaciones sin afectar el código que utiliza esas entidades.**

1. Escriban otro archivo de pruebas, llámelo colonyErrG.txt, para probar la flexibilidad.

Presente las pantallas que contenga un error significativo.

**RETROSPECTIVA**

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes ?(Horas/Hombre)
2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?
3. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?
4. ¿Cuál consideran fue el mayor logro? ¿Por qué?
5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?
6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?