Escuela Colombiana de Ingeniería Julio Garavito

Andrés Cardozo

Tulio Riaño

Laboratorio 06

Programacion Orientada a Objetos

FECHA:

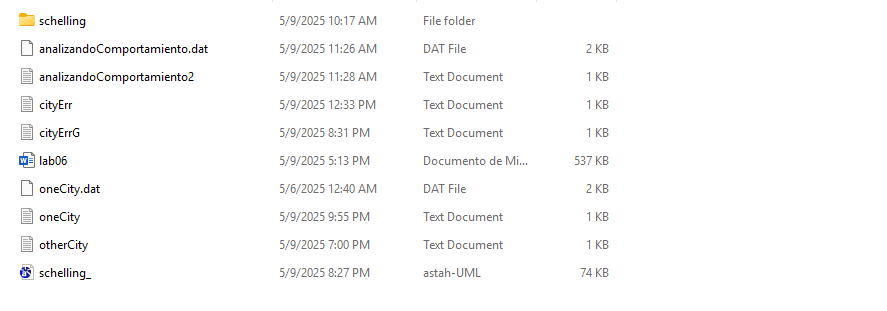
10/05/2025

# **DESARROLLO**

Preparando

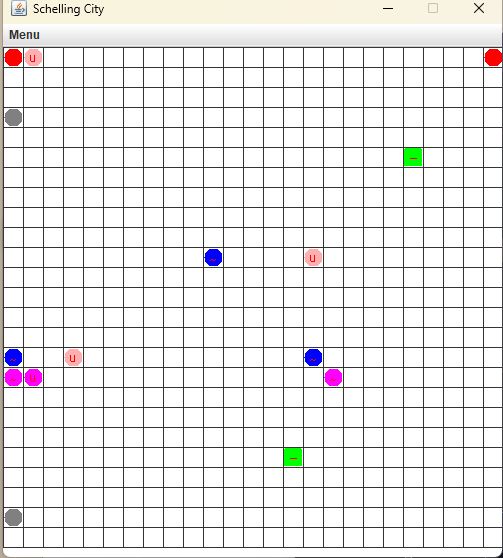
En este laboratorio vamos a extender el proyecto schelling adicionando un menú barra con las opciones básicas de entrada-salida y las opciones estándar nuevo y salir.

1. En su directorio descarguen la versión del proyecto realizado por ustedes para el laboratorio 03 y preparen el ambiente para trabajar desde CONSOLA



1. Ejecuten el programa, revisen la funcionalidad.

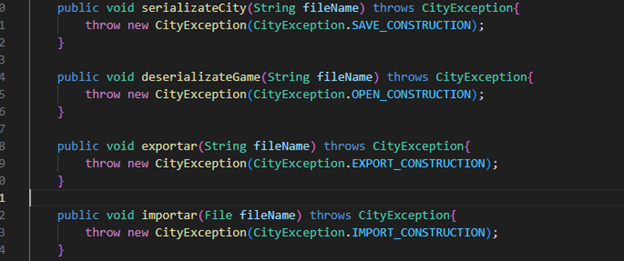
Revisando la funcionalidad del trabajo se observa como se dejo en la entrega del laboratorio 03



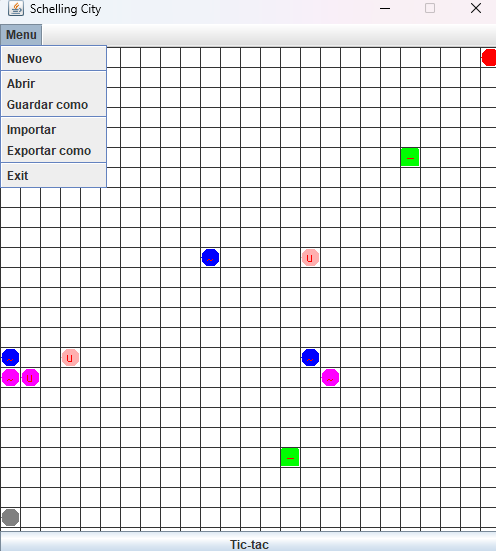
# **Creando la maqueta**

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

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

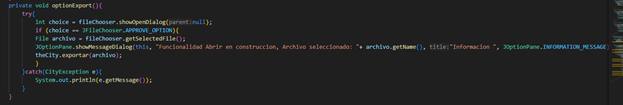


1. VISTA : Construyan un menú barra que ofrezca, además de las opciones básicas de entrada-salida, las opciones estándar de nuevo y salir (Nuevo, Abrir, Guardar como, Importar, Exportar como, Salir). No olviden incluir los separadores. Para esto creen el método prepareElementsMenu. Capturen la pantalla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



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.

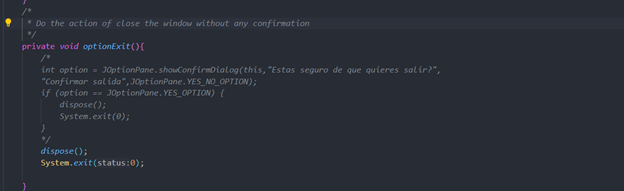




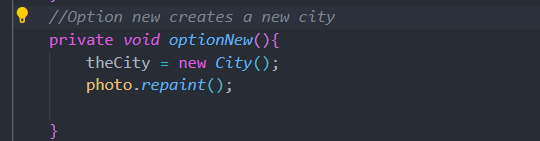
# **Implementando salir y nuevo**

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.

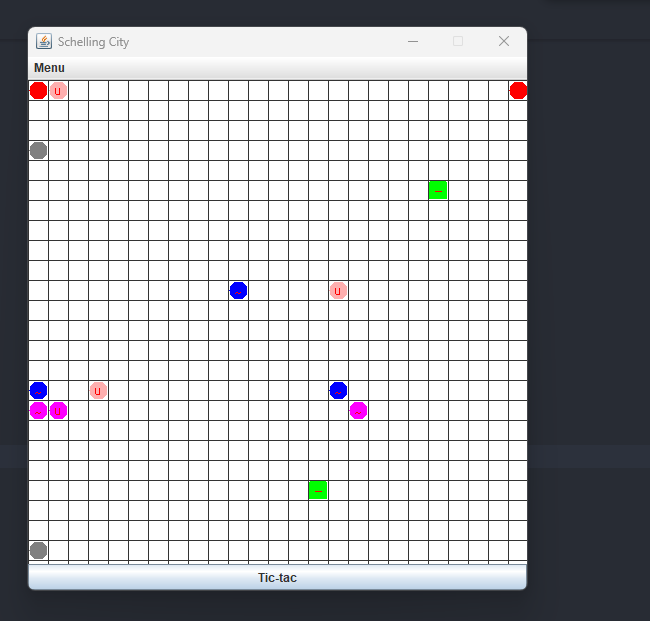
1. Construyan el método optionExit que hace que se termine la aplicación. No es necesario incluir confirmación.

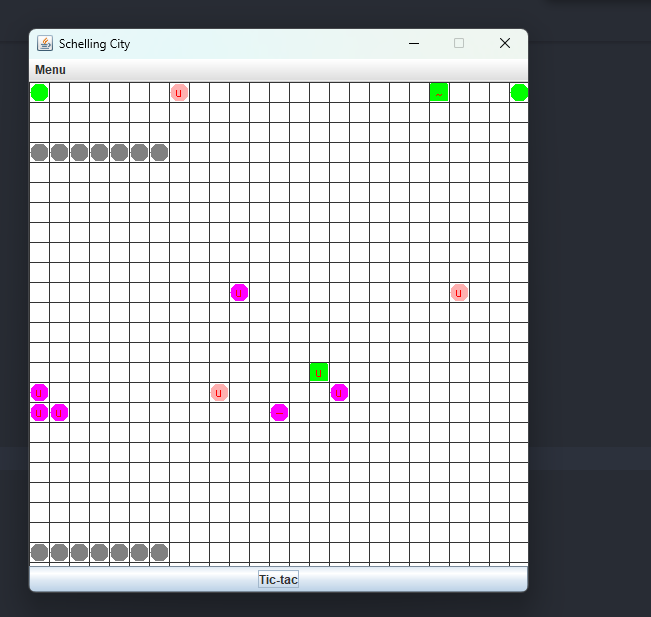


1. Construyan el método optionNew que crea una nueva ciudad. Capturen una pantalla significativa.

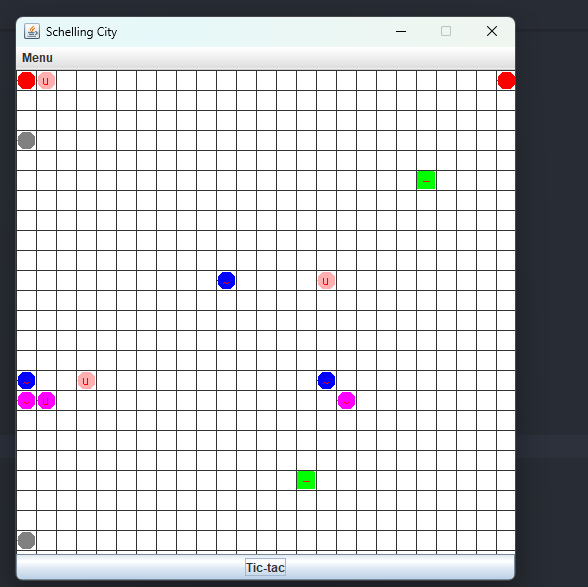


A partir del laboratorio 03 cuando iniciamos la aplicación tenemos un estado inicial de esta forma:

  
Tras aplicar varios tics se obtiene:



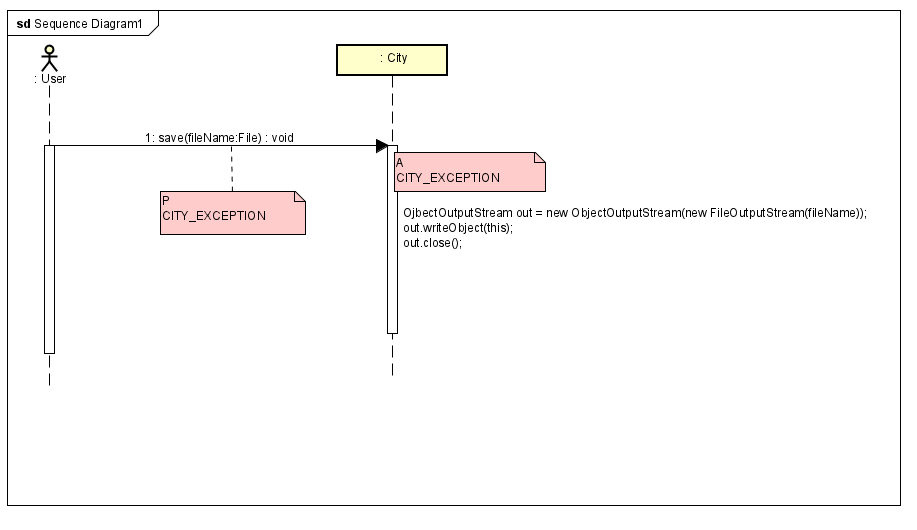
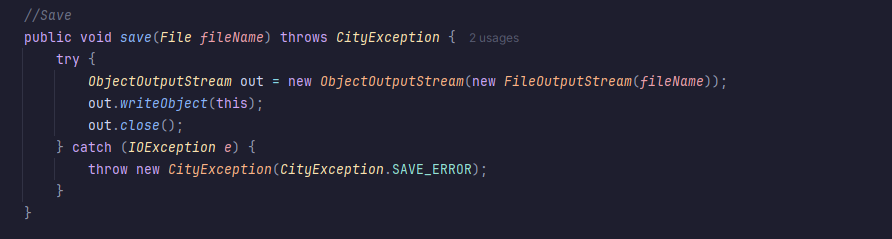
Al aplicar la nueva opción de nueva volveremos al estado original ya que se estaría creando una nueva ciudad.



# **Implementando salvar y abrir**

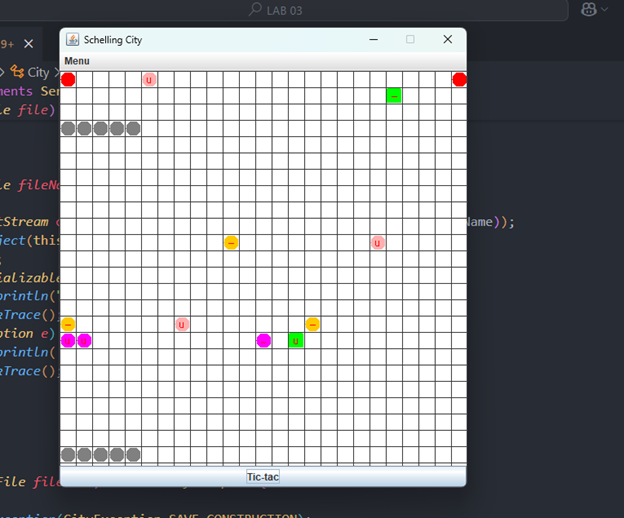
Las opciones salvar y abrir van a ofrecer servicios de persistencia de la ciudad como objeto. Los nombres de los archivos deben tener como extensión .dat.

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 la ciudad. Por ahora para las excepciones sólo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.



1. Validen este método guardando el estado obtenido después de dos clics como oneCity.dat. ¿El archivo se creó en el disco? ¿Cuánto espacio ocupa?

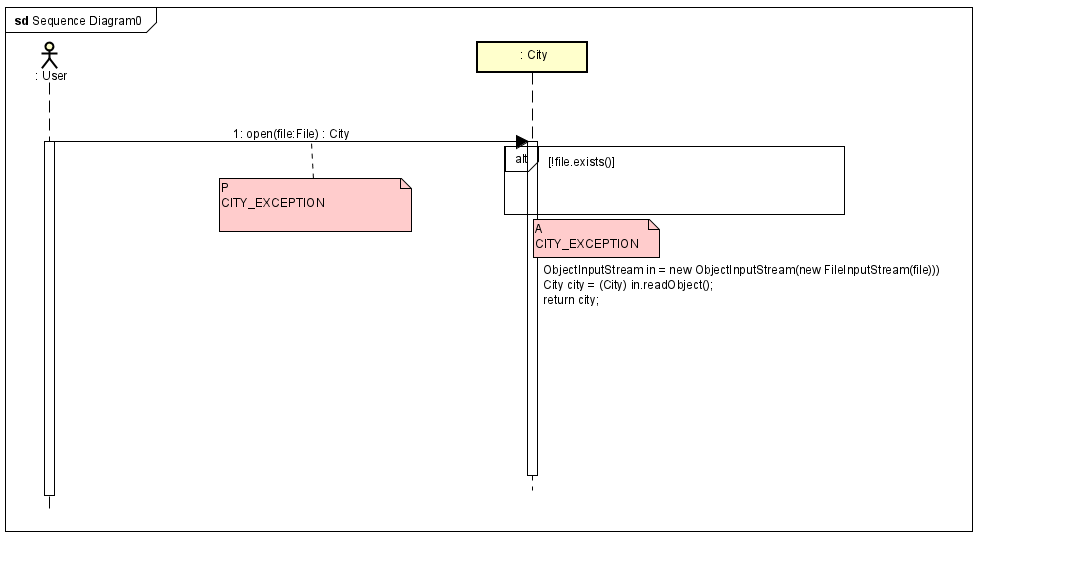
Después de dos clics obtenemos



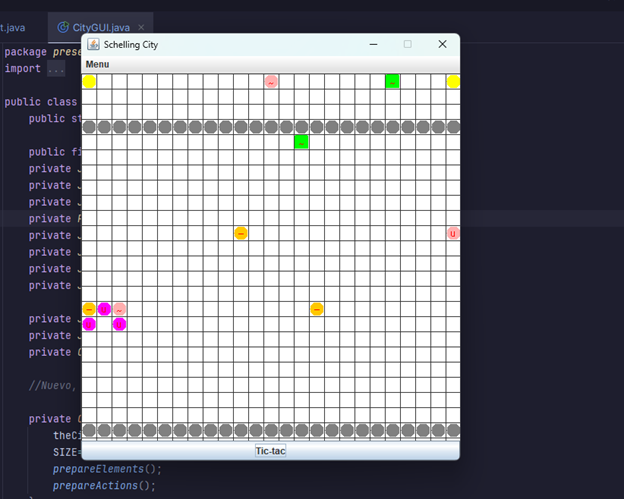
Al utilizar esta nueva funcionalidad se guarda en el disco y pesa 2 KB.



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



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

Iniciando la aplicación al oprimir varios tics llegamos a una situación, como la siguiente  
 

Al abrir el archivo que guardamos anteriormente oneCity.dat:  
 

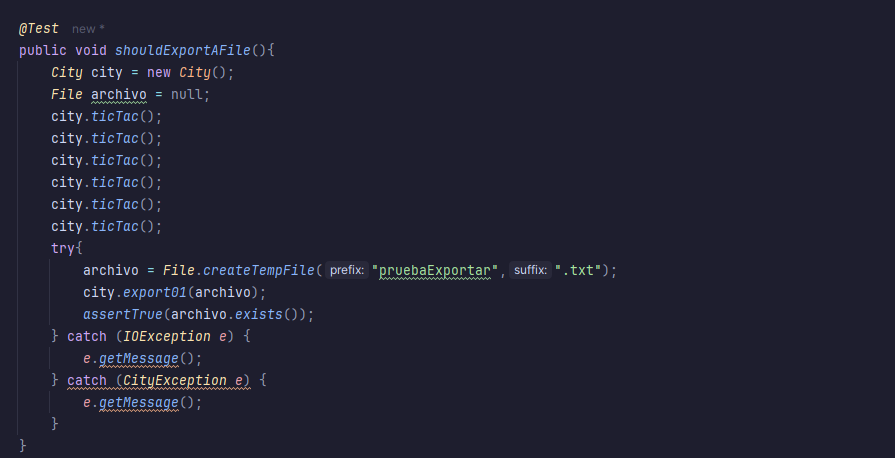
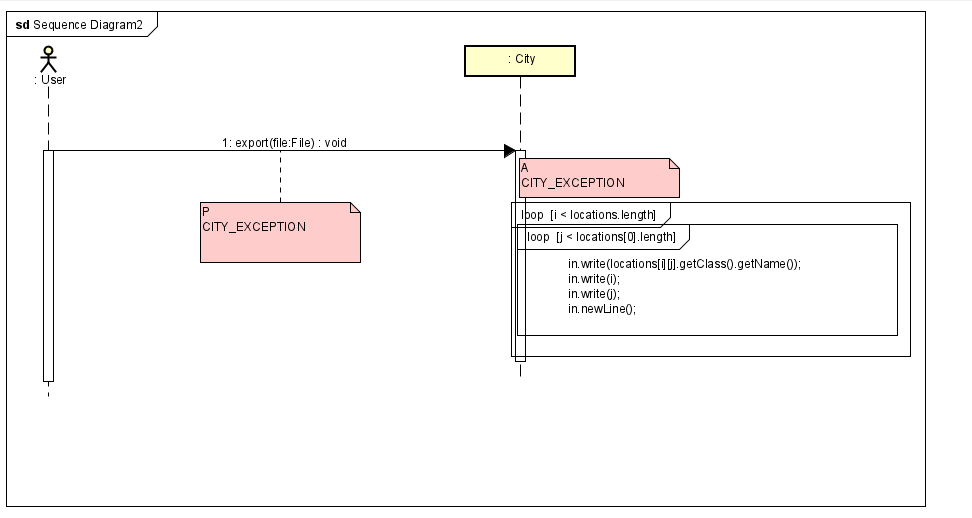
# **Implementando importar y exportar**

Estas operaciones nos van a permitir importar información de la ciudad desde un archivo de texto y exportarla. 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. Person 10 10 Walker 20 20

1. Copien las versiones actuales de import y export y renómbrenlos como import00 y export00

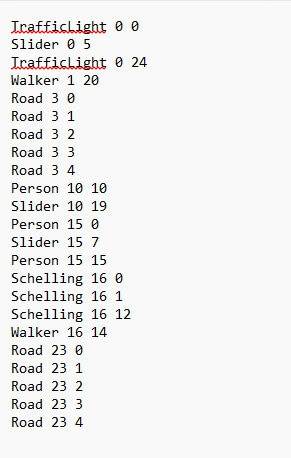
Esto se encuentra en City.java

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 sólo consideren un mensaje de error general. No olviden diseño y pruebas de unidad.

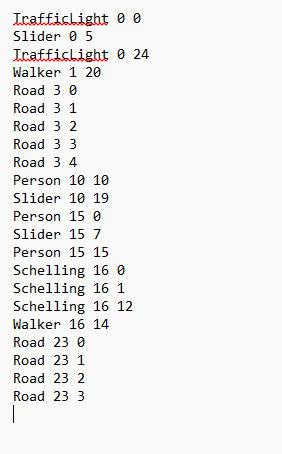


1. Realicen una prueba de aceptación de este método: iniciando la aplicación y exportando como oneCity.txt. Editen el archivo y analicen los resultados. ¿Qué pasó?

Iniciamos la aplicación y hacemos varios tic tacs, luego exportamos el archivo con la extensión oneCity.txt

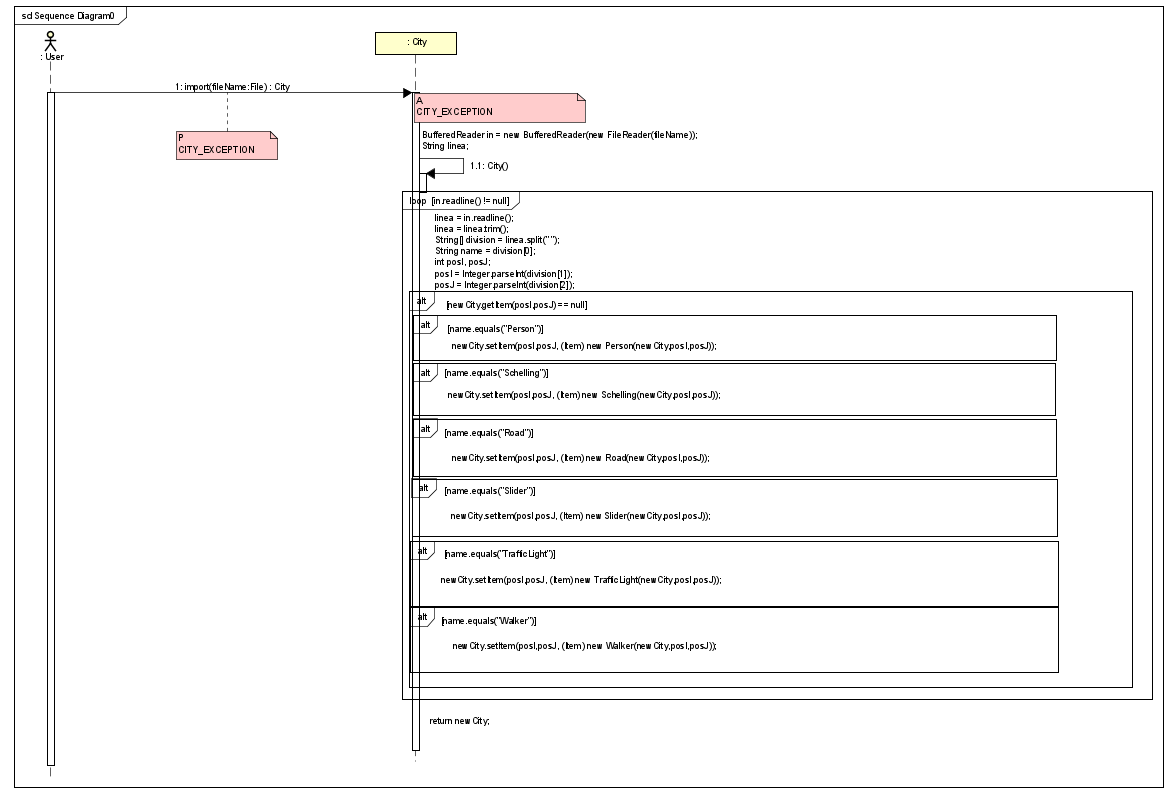
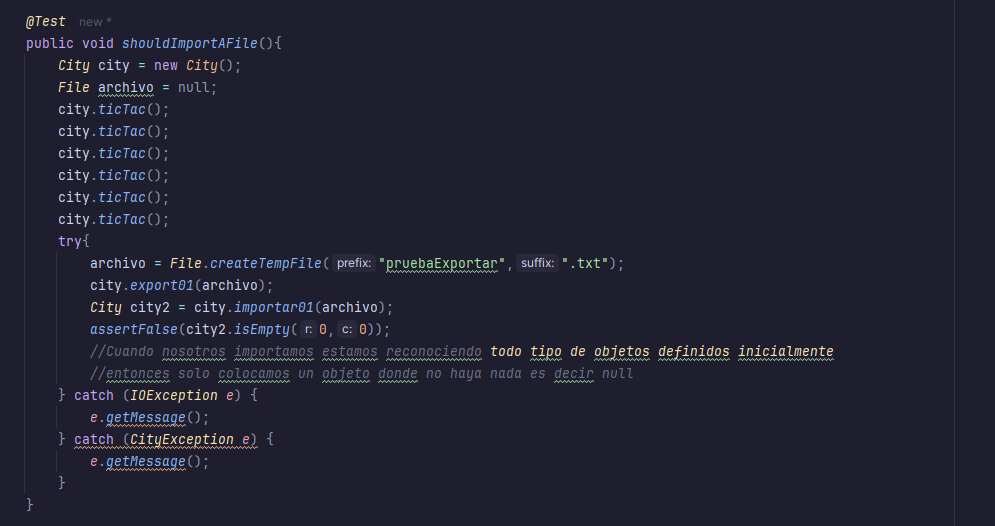
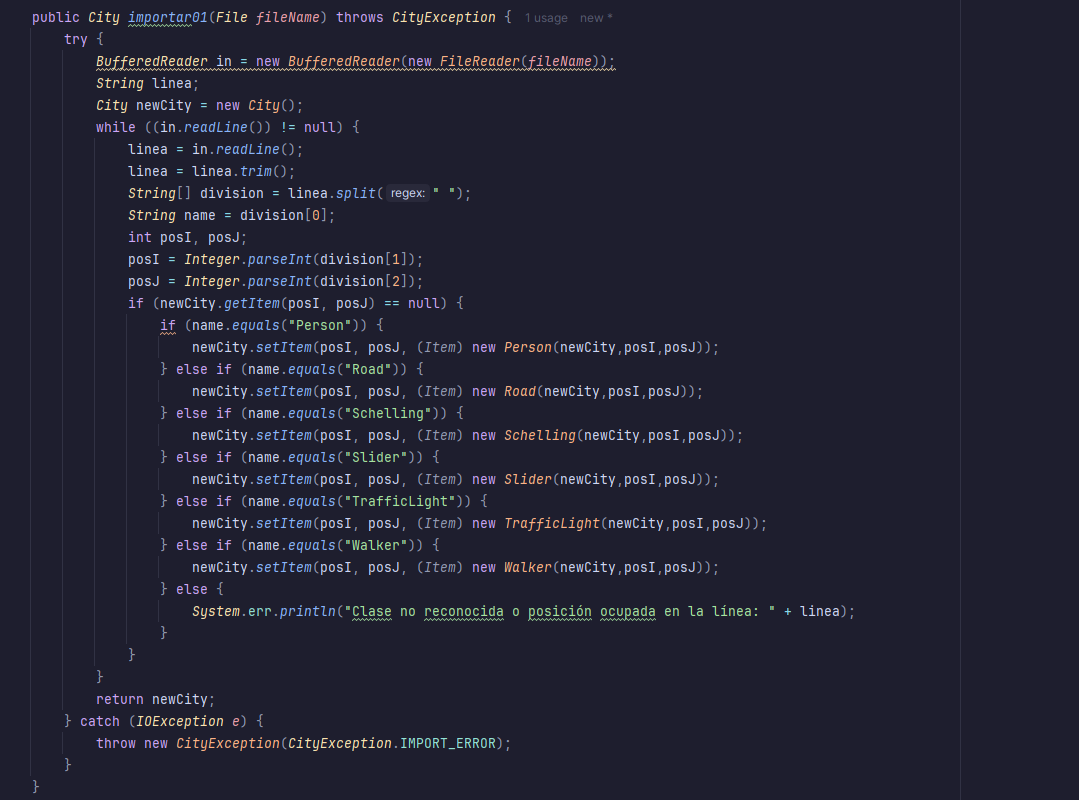


Vamos a editarlo borrando el último elemento.



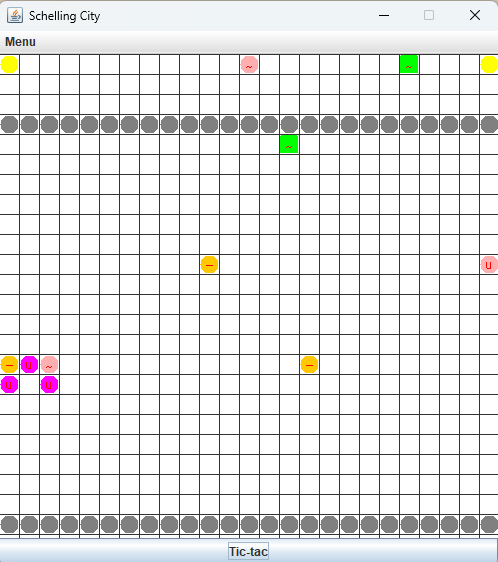
Entonces al editar el archivo solo ponen los objetos existentes sin afectar el proceso de importar. Lo cual seria diferente si serializamos con save, ya que si borramos algo ahi si se daña.

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)



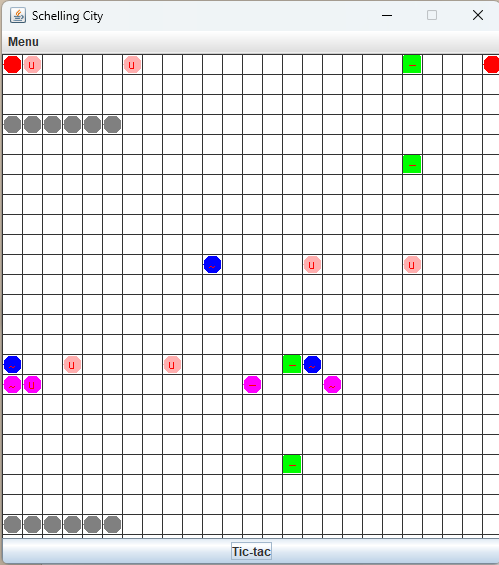
1. Realicen una prueba de aceptación de este par de métodos: iniciando la aplicación exportando a oneCity.txt. saliendo, entrando, creando una nueva e importando el archivo otherCity.txt. ¿Qué resultado obtuvieron? Capturen la pantalla final.

Al tener una estado inicial de esta forma:



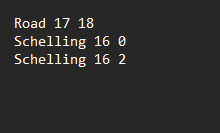
Saliendo de la aplicación y creando una nueva e importando otherCity.txt obtenemos

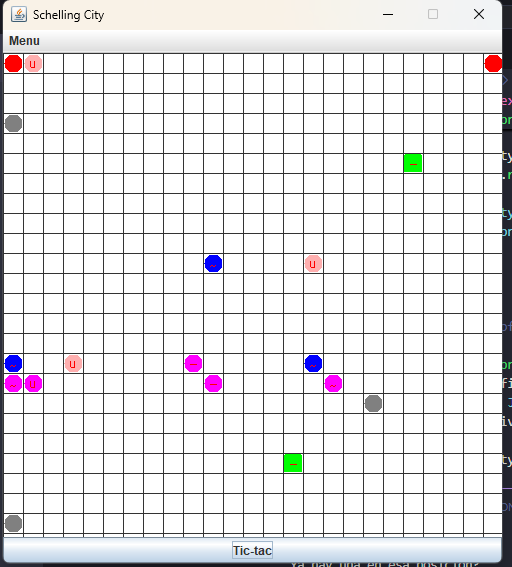
Obtenemos la ciudad creada en otherCity.txt:



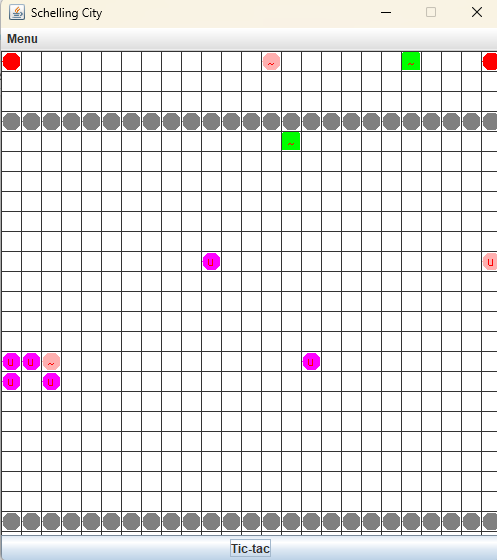
1. Realicen otra prueba de aceptación de este método escribiendo un archivo de texto correcto en oneCity.txt. e importe este archivo. ¿Qué resultado obtuvieron? Capturen la pantalla.

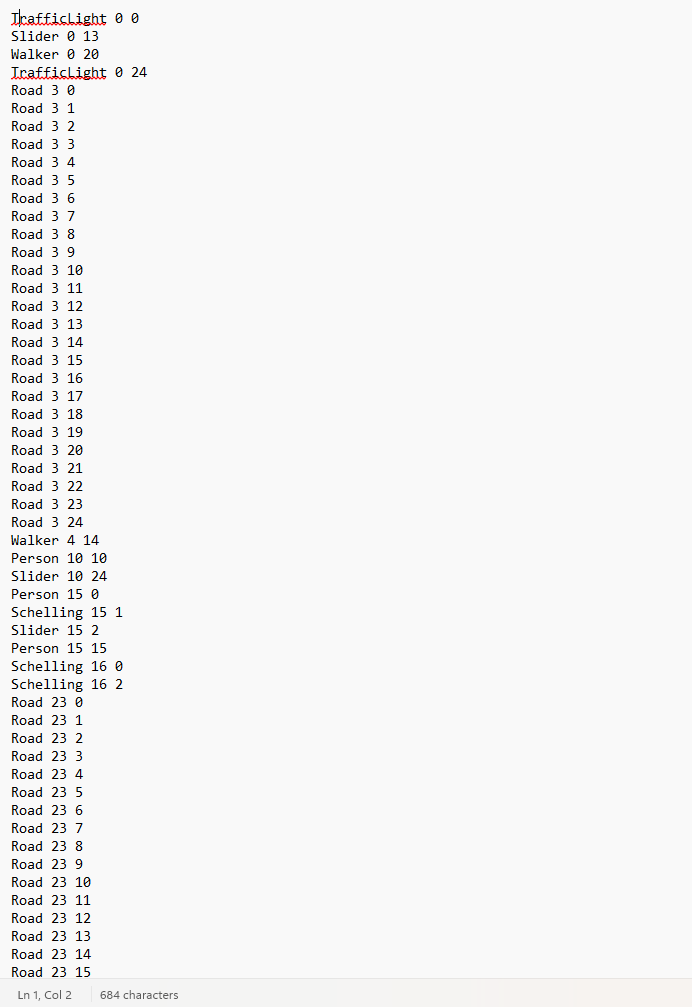
Creando uno a mano:

  
E intentando importarlo:

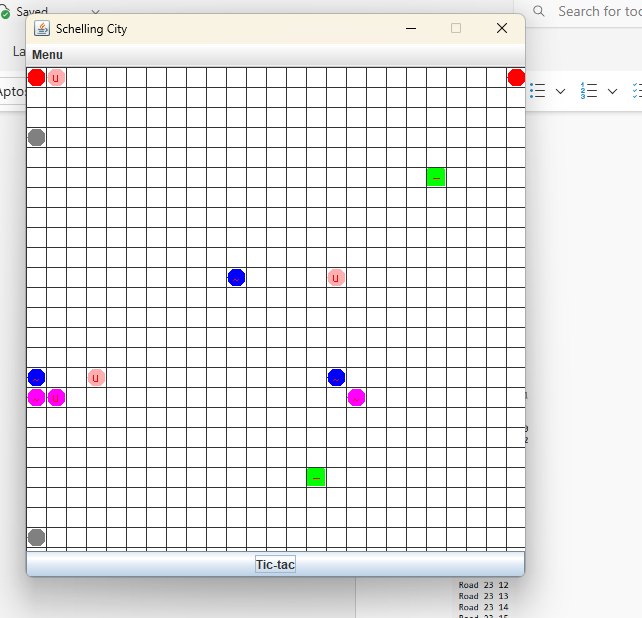


Iniciando la aplicación y teniendo varios tics tacs tenemos un estado:

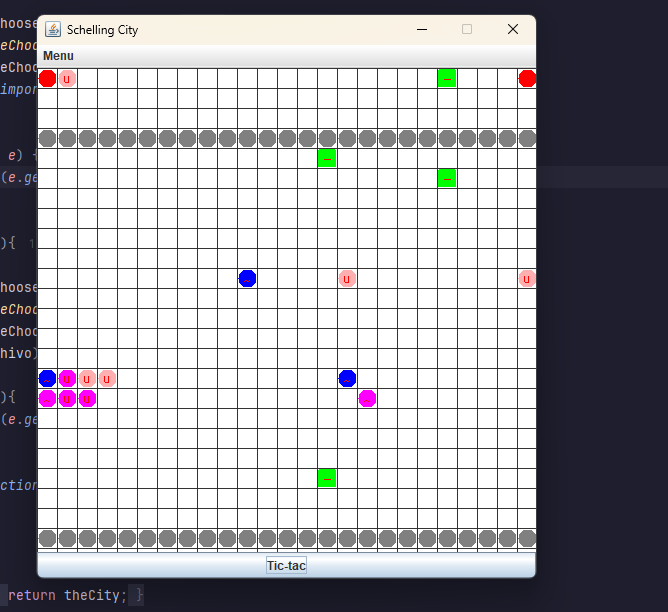




Al crear nuevamente un nuevo archivo



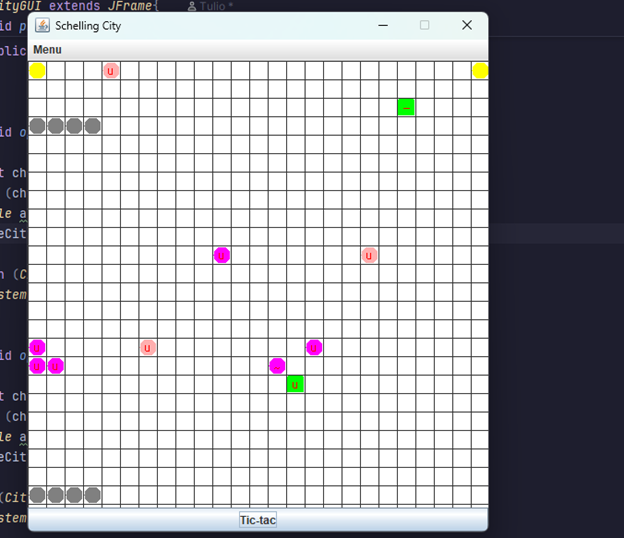
E importar oneCity.txt obtenemos el resultado correcto:



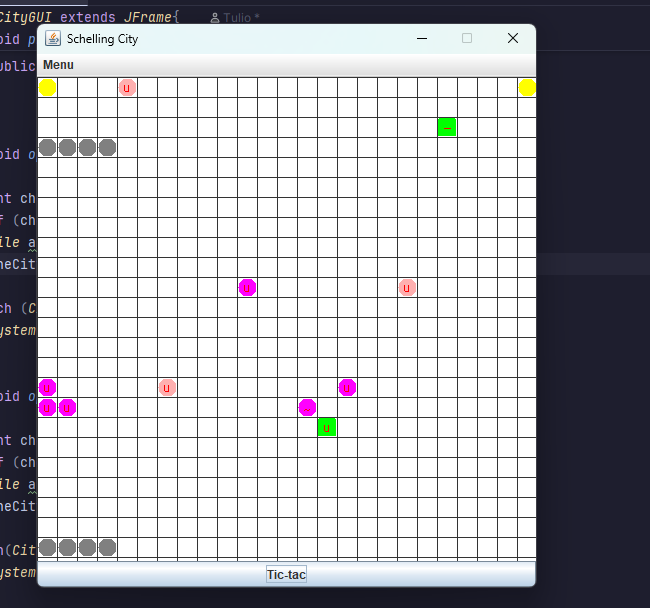
# **Analizando comportamiento**

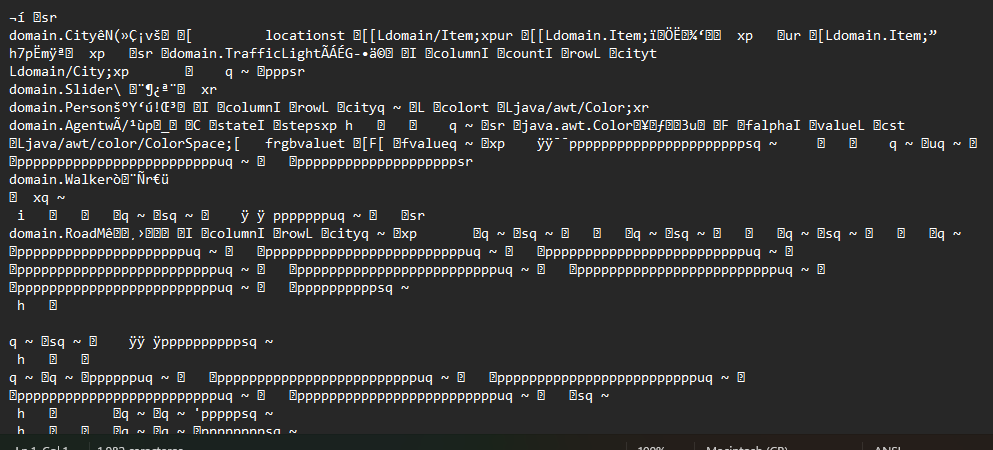
1. Ejecuten la aplicación, den tres clics, salven a un archivo cualquiera y ábranlo. Describan el comportamiento

Tras ejecutar la aplicación y dar tres clics obtenemos el siguiente resultado:



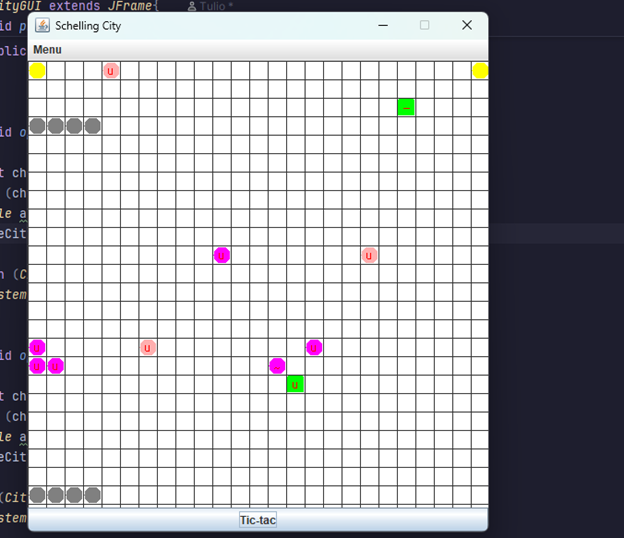
Creando un nueva ciudad y al utilizar esta nueva funcionalidad de abrir obtenemos el resultado esperado:

  
por lo que esta funcionalidad funciona de manera correcta.

  
Encontramos el archivo serializado el cual convierte en bytes los objetos y para mas adelante que la maquina lo entienda.

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

Nuevamente al ejecutar la aplicación y dar tres clics obtenemos el mismo escenario del punto anterior al exportar verificamos el archivo txt que tenga escrita correctamente las posiciones.





Cuando importamos obtenemos el resultado esperado, pero cuando estamos realizando esta funcionalidad estamos sobreescribiendo sobre la ciudad por defecto por lo que hay posiciones que van a ser iguales(someItems).

Por lo que quitamos mas adelante los items defaults.

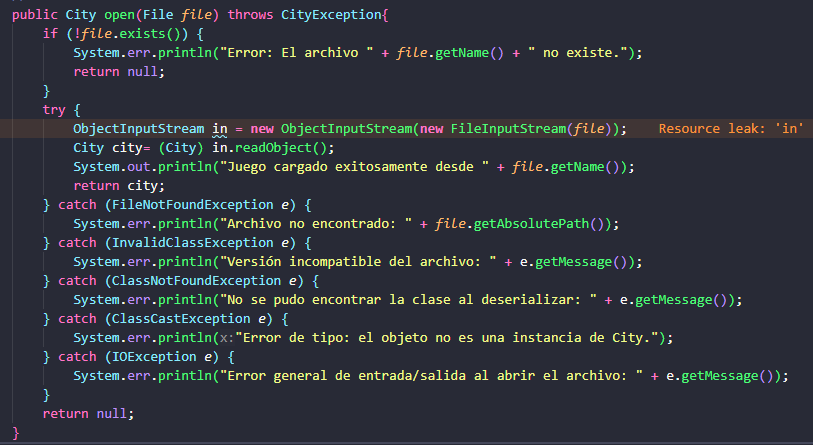
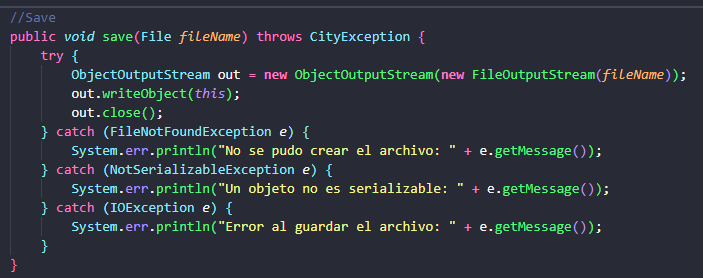
1. ¿Qué diferencias ven el comportamiento 1. y 2.? Expliquen los resultados.

Cuando abrimos un archivo .dat la ciudad se abre y no se sobreescribe sobre ninguna otra ciudad, mientras que cuando importamos sobreescribimos sobre la ciudad por defecto. Otra de las diferencias es el tamaño de los archivos cuando guardamos un archivo .dat pesa 2 KB, mientras que al exportar un archivo .txt este pesa 1 KB. Por lo que es bueno saber cuando usar una forma u otra.

# **Perfeccionando salvar y abrir**

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. No olviden pruebas de unidad.

Para estos métodos manejamos excepciones más explicitas sobre que podría llegar a ocurrir al utilizar este tipo de funcionalidades.



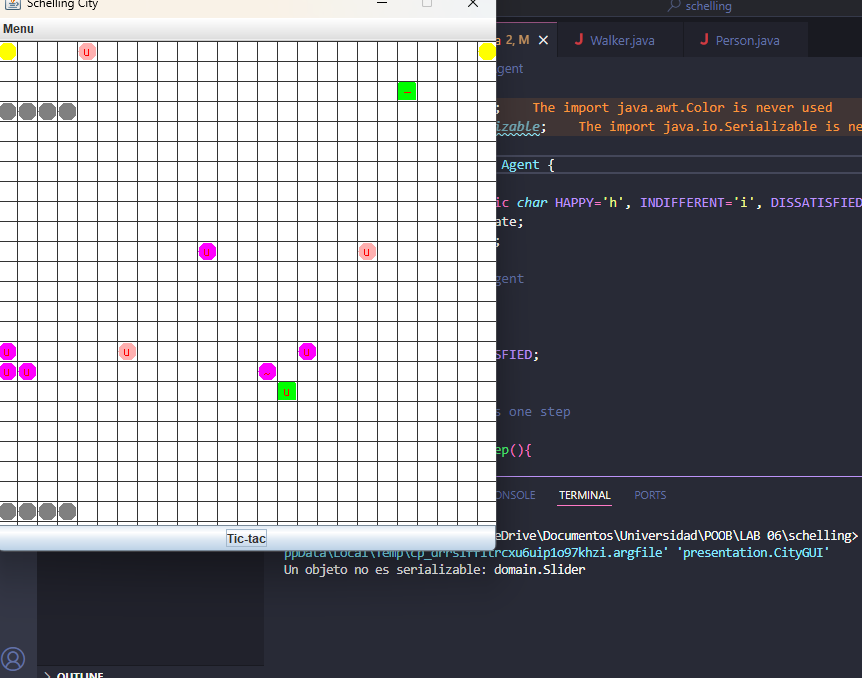
Por ejemplo que el archivo no exista.

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

Open:

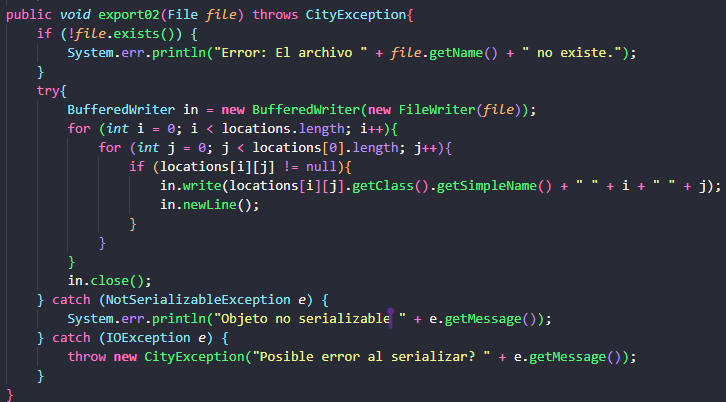
Al tratar de abrir un archivo inexistente, se llega a observar este tipo de mensajes.



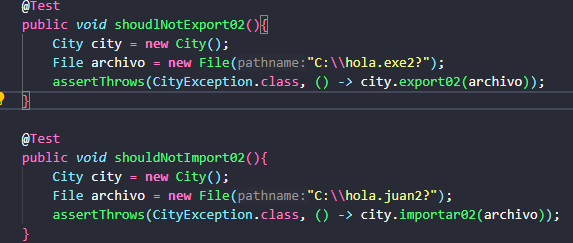
Save: 

# **Perfeccionando importar y exportar.**

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. No olviden pruebas de unidad.

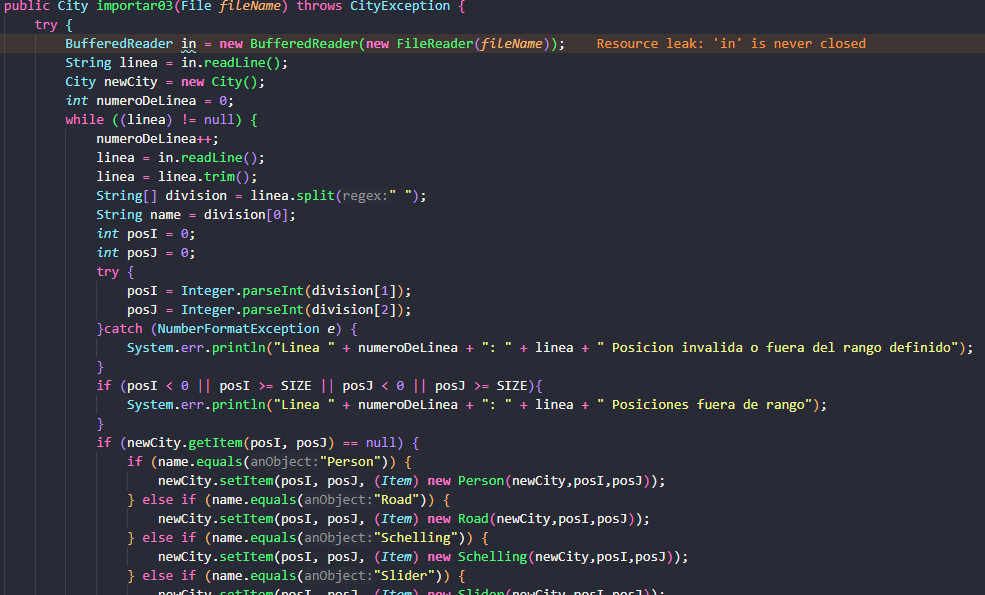


1. Realicen una prueba de aceptación para validar uno de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.



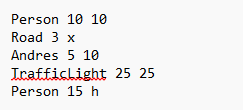
# **Perfeccionando importar. Hacia un minicompilador.**

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, palabra que tiene el error y causa de error.

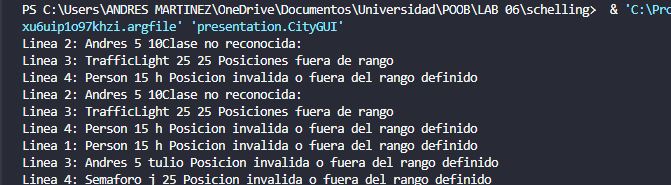


1. Escriban otro archivo con errores, llámelo cityErr.txt, para ir arreglándolo con ayuda de su “importador”. Presente las pantallas que contengan los errores.

Creamos un archivo llamado cityErr.txt que contiene la siguiente información



Al ejecutar la aplicación podemos ver que funciona el compilador

Se identifican el error que es fuera de rango ya que las posiciones que importamos son mayores al tamaño de la cuadricula.

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

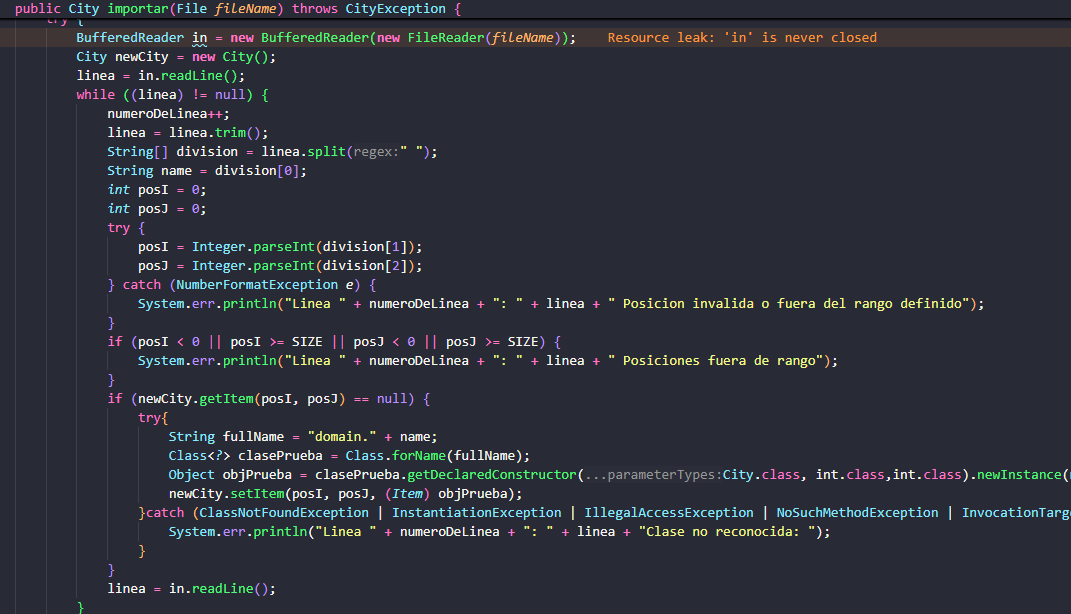
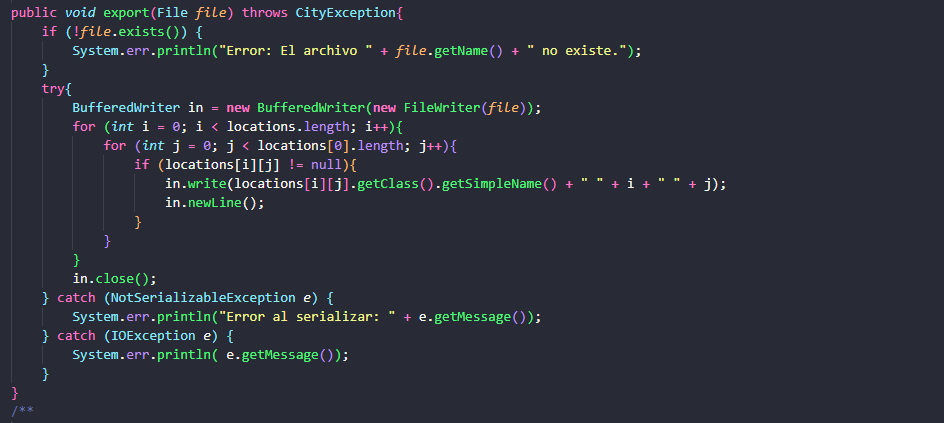
1. Copien las versiones actuales de import y export y renómbrenlos como import03 y export03

Esto se evidencia en City.java

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)

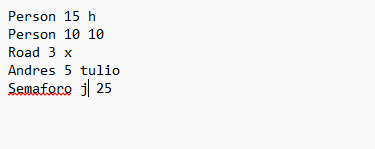
Mediante la reflexión en java se puede identificar a que clase pertenece el objeto.

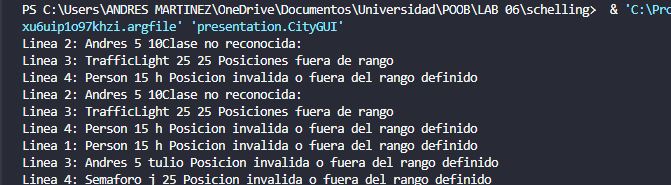
Luego lo que tenemos que hacer es corregir la implementación que hicimos en import02 e import03 aplicando reflexión.

  
Vemos que ya no tenemos los if por cada clase. Que no era extensible.

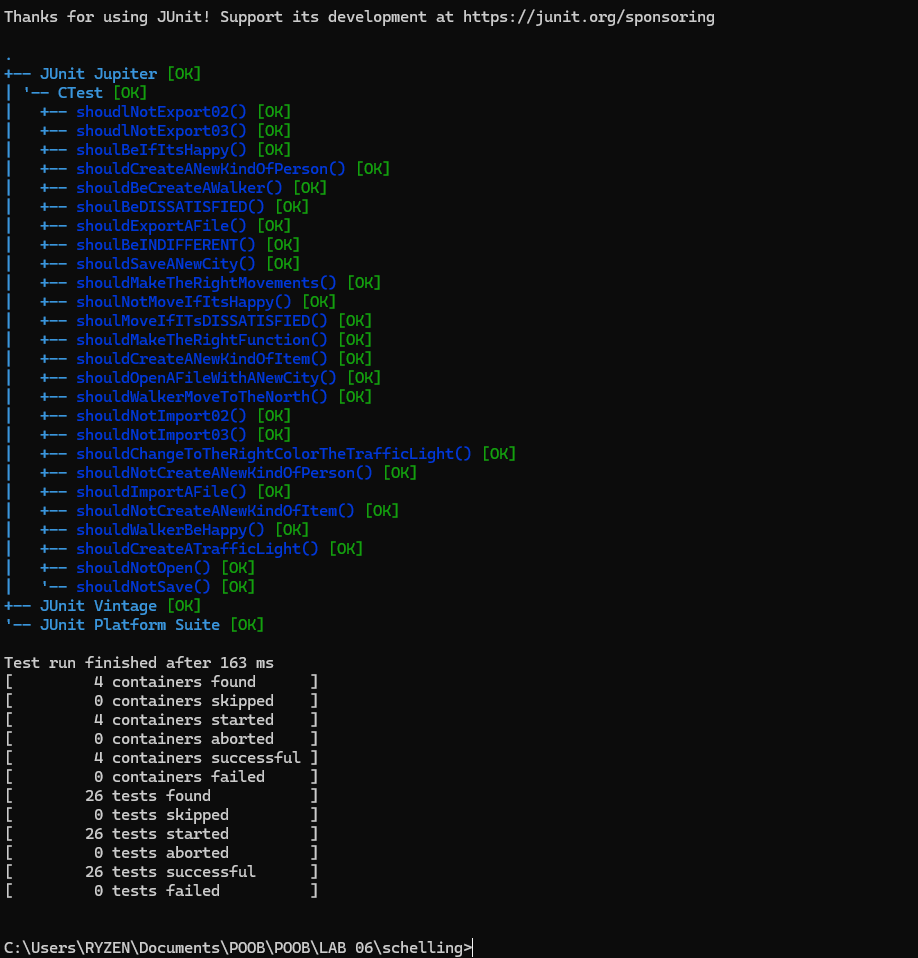
1. Escriban otro archivo de pruebas, llámelo cityErrG.txt, para probar la flexibilidad. Presente las pantallas que contenga un error significativo.

Este nuevo archivo de cityErrG.txt contiene nuevamente errores parecidos al anterior pero más enfocados en la identificación de clases.



La clase llamada andrés no esta definida en schelling ni una posición diferente a un entero por lo que se puede evidenciar este error al importar el archivo.

**PRUEBAS EJECUTADAS POR CONSOLA**



# Retrospectiva

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

Hombre)

Andrés Cardozo: 6 Horas

Tulio Riaño: 6 Horas

2. ¿Cuál es el estado actual de laboratorio? ¿Por qué?

El estado actual del laboratorio es completo, se desarrollaron todos los ciclos incluyendo los bonos de manera acorde y correcta.

3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son

Importante?

Consideramos que las prácticas XP presentadas para este laboratorio son importantes ya que cuando se llega a presentar un Bug se puede hacer una prueba para identificar este, por otra parte la integración del código a la vez mediante la coordinación y la comunicación puede llegar a hacer que el trabajo sea más eficiente y agradable.

4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor

problema? ¿Qué hicieron para resolverlo?

Nuestro mayor logro fue realizar de manera correcta la importación, este también fue un problema ya que a veces se nos presentaban dificultades a la hora de leer de manera correcta el archivo e importarlo a la cuadricula de City.

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

Resultados?

De nuevo resaltamos la comunicación y disposición para desarrollar el laboratorio, así mismo mantenemos nuestro compromiso por la ayuda mutua que se evidencia en el desempeño.

6. ¿Qué referencias usaron? ¿Cuál fue la más útil?

La mayoría de las referencias que se adjuntaron fueron demasiado útil a la hora de implementar el desarrollo correspondiente a la interfaz.

[Core Java Reflection](https://docs.oracle.com/javase/8/docs/technotes/guides/reflection/index.html)