PROGRAMACIÓN ORIENTADA A OBJETOS

Persistencia 2025-01

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.

   Use collective ownership.

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.

DESARROLLO

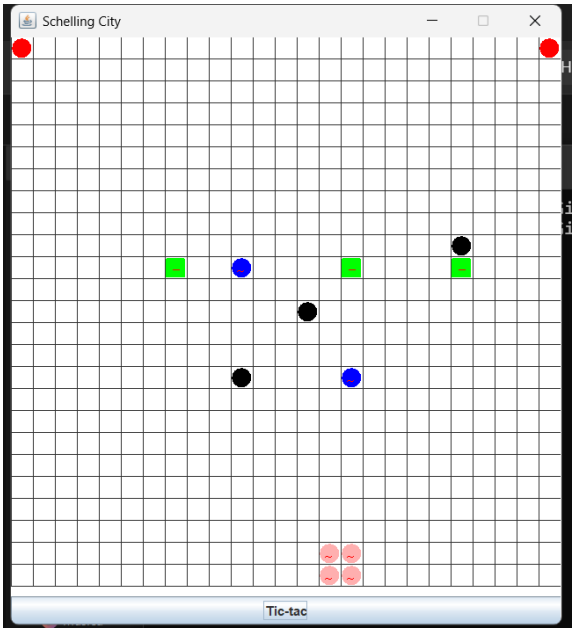
Preparando

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**

```
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> New-Item -ItemType Directory -Path ".\bin\domain", ".\bin\presentatio
n"


    Directorio: C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling\bin


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----         3/05/2025      7:29                domain
d-----         3/05/2025      7:29                presentation


PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> Move-Item -Path ".\presentation\*.class" -Destination ".\bin\presenta
tion\"
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> Move-Item -Path ".\domain\*.class" -Destination ".\bin\domain\"
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> Move-Item -Path ".\domain\*.java" -Destination ".\src\domain\"
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> Move-Item -Path ".\presentation\*.java" -Destination ".\src\presentat
ion\"
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling> Move-Item -Path ".\presentation\*.class" -Destination ".\bin\presenta
tion\"
PS C:\Users\Karol\OneDrive\Documentos\GitHub\EntregasPoob\lab6\schelling>
```

2. Ejecuten el programa, revisen la funcionalidad.
   Funciona correctamente.



Creando la maqueta

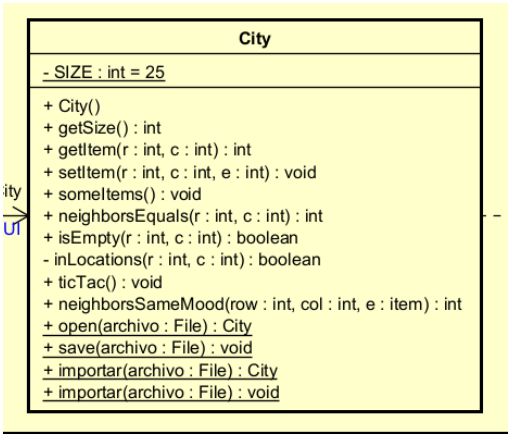
[En lab06.doc, *.asta y *.java] **[NO OLVIDEN BDD y MDD]**

En este punto vamos a construir la maqueta correspondiente a esta extensión

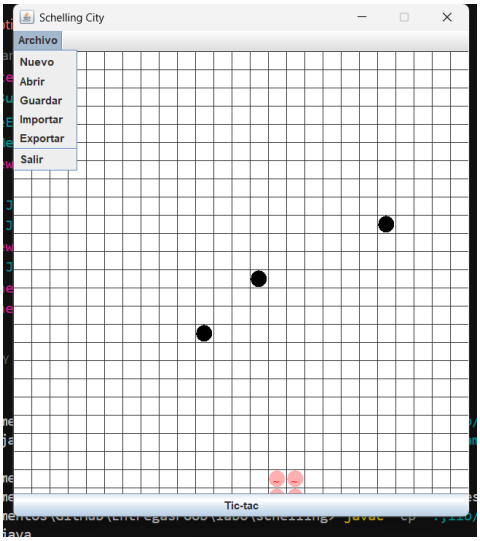
siguiendo el patrón MVC.

1. **MODELO:** Preparen en la clase fachada del 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.

```
public static  City open (File archivo) throws CityException{

    throw new CityException(CityException.ERROR_ABRIR + archivo.getName());
}


public void save (File archivo) throws CityException{

    throw new CityException(CityException.ERROR_GUARDAR + archivo.getName());
}


public static City importar (File archivo) throws CityException{

    throw new CityException(CityException.ERROR_IMPORTAR + archivo.getName());
}


public void exportar (File archivo) throws CityException{

    throw new CityException(CityException.ERROR_EXPORTAR + archivo.getName());
}
```

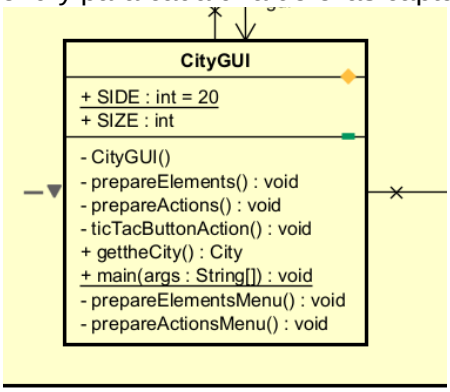


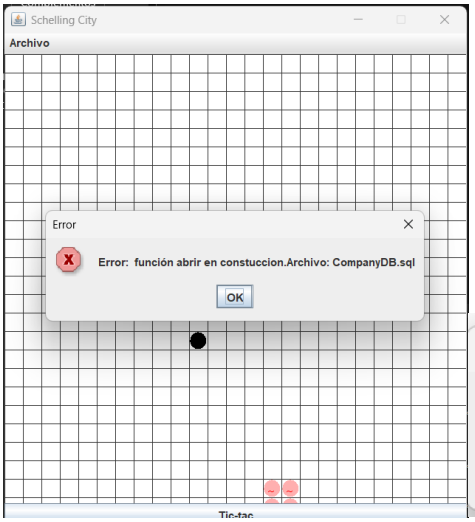
2. **VISTA :** Construyan un menú barra que ofrezca, además de las opciones básicas de entrada-salida, las opciones estándar de nuevo y salir (Nuevo, Abrir, Guardar como, Importar, Exportar como, Salir). No olviden incluir los separadores. Para esto creen el método prepareElementsMenu. Capturen la pantalla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

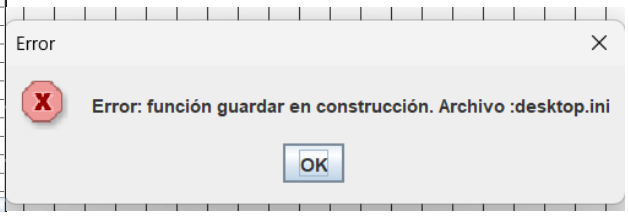


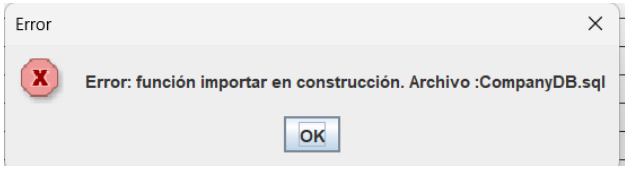
```
*/
private void prepareElementsMenu() {
    menuBar = new JMenuBar();
    menuArchivo = new JMenu(s:"Archivo");

    itemNuevo = new JMenuItem(text:"Nuevo");
    itemAbrir = new JMenuItem(text:"Abrir");
    itemGuardar = new JMenuItem(text:"Guardar");
    itemSalir = new JMenuItem(text:"Salir");
    itemImportar = new JMenuItem(text:"Importar");
    itemExportar = new JMenuItem(text:"Exportar");

    menuArchivo.add(itemNuevo);
    menuArchivo.add(itemAbrir);
    menuArchivo.add(itemGuardar);
    menuArchivo.add(itemImportar);
    menuArchivo.add(itemExportar);

    menuArchivo.add(new JSeparator());
    menuArchivo.add(itemSalir);

    menuBar.add(menuArchivo);
    setJMenuBar(menuBar);
}
```

3. **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.




Schelling City
Archivo
Error
Error: función abrir en constuccion.Archivo: CompanyDB.sql
OK


Error
Error: función guardar en construcción. Archivo :desktop.ini
OK


Error
Error: función importar en construcción. Archivo :CompanyDB.sql
OK


Error
Error: función exportar en construcción. Archivo :desktop.ini
OK

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.

```
    itemSalir.addActionListener(s -> {
        optionExit();
    });
}

private void optionExit(){
    System.exit(status:0);
}
```

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

```
/**
 * metodo para empezar una nueva ciudad
 */
private void optionNew() {
int confirm = JOptionPane.showConfirmDialog(
    this,
    message:"¿Estás seguro de que deseas crear una nueva ciudad? Se perderán los cambios no guardados.",
    title:"Confirmar",
    JOptionPane.YES_NO_OPTION
);

if (confirm == JOptionPane.YES_OPTION) {
    theCity = new City();
    photo.repaint();
    JOptionPane.showMessageDialog(this, message:"Nueva ciudad creada exitosamente.");
}
```

Implementando salvar y abrir

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

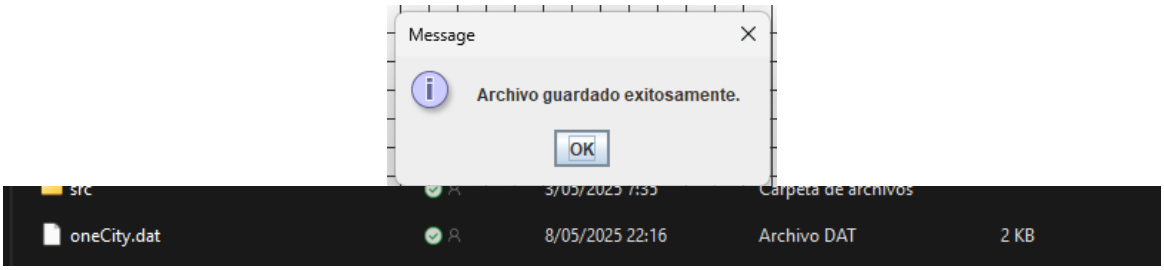
Las opciones salvar y abrir van a ofrecer servicios de persistencia de la ciudad como objeto. Los nombres de los archivos deben tener como extensión .dat.

1. Copien las versiones actuales de open y savey 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.

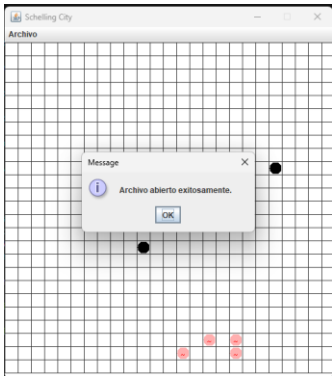
```
/**
 * implementacion de save con serializacion
 * @param archivo
 * @return City guardada
 * @throws CityException
 */
public void save(File archivo) throws CityException {
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(archivo))) {
        oos.writeObject(this);
    } catch (IOException e) {
        throw new CityException(CityException.ERROR_GUARDAR + archivo.getName());
    }
}
```

```
    @Test
public void shouldSaveCityToFile() {
    try {
        File archivo = new File(pathname:"oneCity.dat");
        ciudad.save(archivo);
        assertTrue(message:"El archivo no fue creado.", archivo.exists());
        assertTrue(message:"El archivo está vacío.", archivo.length() > 0);
        archivo.delete();
    } catch (CityException e) {
        fail("No se esperaba excepción al guardar: " + e.getMessage());
    }
}
```

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



4. Construyan el método openque 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.



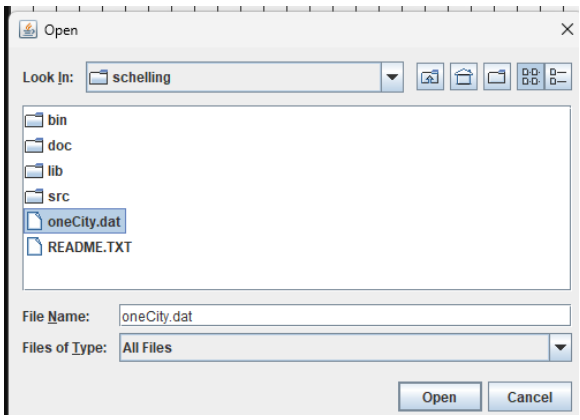

```java
public static City open(File archivo) throws CityException {
    try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(archivo))) {
        return (City) ois.readObject();
    } catch (IOException | ClassNotFoundException e) {
        throw new CityException(CityException.ERROR_ABRIR + archivo.getName());
    }
}
```

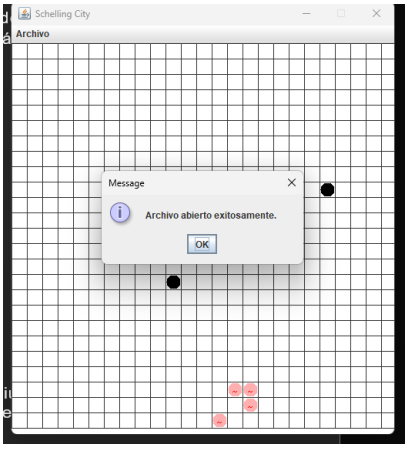
```
JUnit version 4.13.2
..............
Time: 0,11

OK (14 tests)
```

pruebas de unidad de open y sabe

```java
@Test
public void shouldOpenCityFromFile() {
    try {

        ciudad.setItem(r:5, c:5, new Solitaria(ciudad, row:5, col:5));
        File archivo = new File(pathname:"oneCity.dat");
        ciudad.save(archivo);

        City ciudadCargada = City.open(archivo);
        assertTrue(message:"El item no fue restaurado correctamente", ciudadCargada.getItem(r:5, c:5) != null);
        assertTrue(message:"El tipo de objeto no coincide", Solitaria.class.equals(ciudadCargada.getItem(r:5, c:5).getClass()));
        archivo.delete();
    } catch (CityException e) {
        fail("No se esperaba excepción al abrir: " + e.getMessage());
    }
}
```

5. 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.





Implementando importar y exportar

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

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

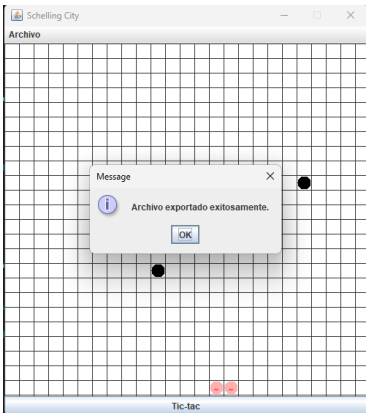
En cada línea asociada un elemento se especifica el tipo y la posición. Person 10 10

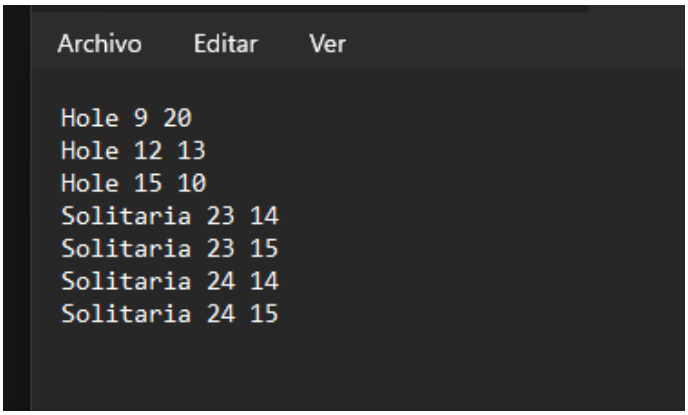
Walker 20 20

1. Copien las versiones actuales de import y exporty renómbrenlos como import00y 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.

```java
@Test
public void shouldExportCityToTextFile() {
    try {
        File archivo = new File(pathname:"cityTestExport.txt");

        // Poner algunos elementos en la ciudad
        ciudad.setItem(r:5, c:5, new Person(ciudad, row:5, column:5));
        ciudad.setItem(r:10, c:10, new Walker(ciudad, row:10, column:10));
        ciudad.setItem(r:15, c:15, new Hole(ciudad, row:15, column:15));
        ciudad.setItem(r:20, c:20, new Solitaria(ciudad, row:20, col:20));

        ciudad.exportar(archivo);
        assertTrue(archivo.exists());
        assertTrue(archivo.length() > 0);

        // Leer archivo para verificar contenido
        BufferedReader reader = new BufferedReader(new FileReader(archivo));
        String line;
        boolean foundPerson = false, foundWalker = false, foundHole = false, foundSolitaria = false;
        while ((line = reader.readLine()) != null) {
            if (line.equals(anObject:"Person 5 5")) foundPerson = true;
            if (line.equals(anObject:"Walker 10 10")) foundWalker = true;
            if (line.equals(anObject:"Hole 15 15")) foundHole = true;
            if (line.equals(anObject:"Solitaria 20 20")) foundSolitaria = true;
        }
        reader.close();
        archivo.delete();

        assertTrue(foundPerson && foundWalker && foundHole && foundSolitaria);
    } catch (Exception e) {
        fail("No se esperaba excepción: " + e.getMessage());
    }
}
```

3. 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?



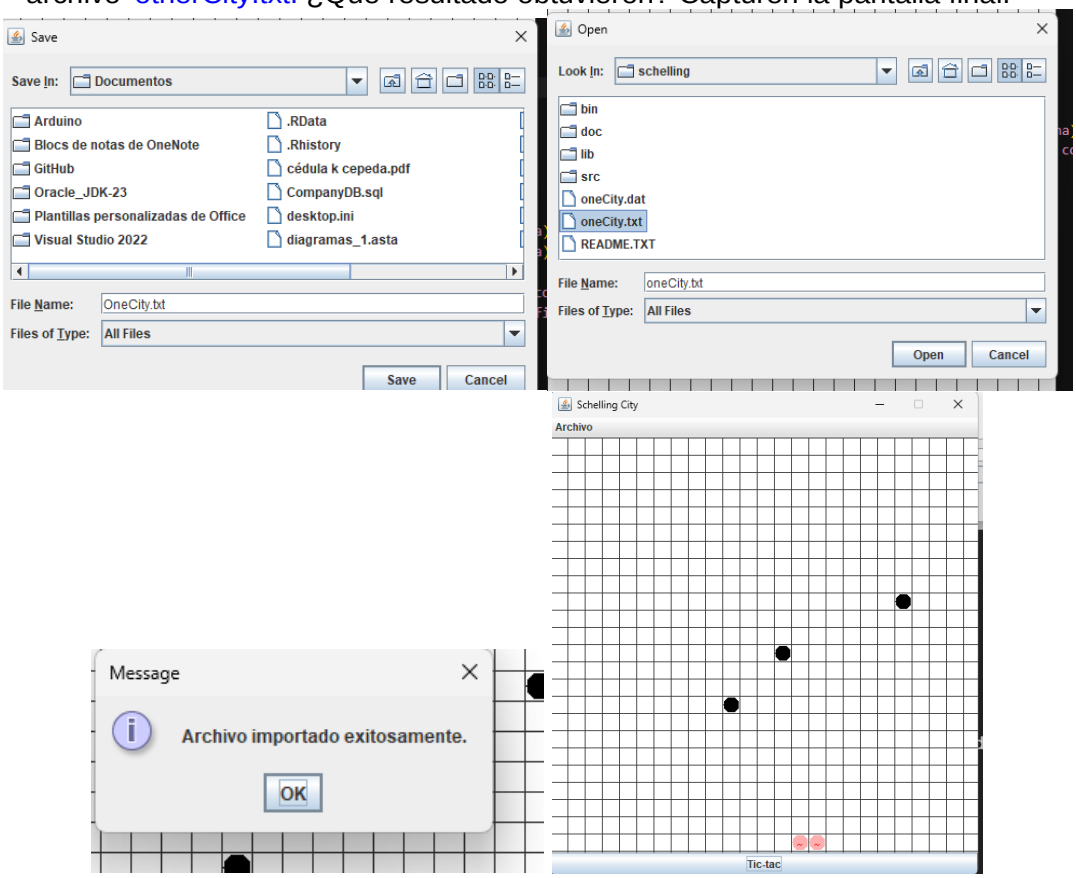


A pesar de modificar el archivo este no cambia el juego actual y debido a que no se ha implementado importar todavía, no se puede abrir el archivo con open.

4. Construyan el método importque 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)

```java
@Test
public void shouldImportCityFromTextFile() {
    try {
        File archivo = new File(pathname:"cityTestImport.txt");

        PrintWriter writer = new PrintWriter(archivo);
        writer.println(x:"Person 6 6");
        writer.println(x:"Walker 7 7");
        writer.println(x:"Hole 8 8");
        writer.println(x:"Solitaria 9 9");
        writer.close();

        City nuevaCiudad = City.importar(archivo);
        archivo.delete();

        assertTrue(nuevaCiudad.getItem(r:6, c:6) instanceof Person);
        assertTrue(nuevaCiudad.getItem(r:7, c:7) instanceof Walker);
        assertTrue(nuevaCiudad.getItem(r:8, c:8) instanceof Hole);
        assertTrue(nuevaCiudad.getItem(r:9, c:9) instanceof Solitaria);
    } catch (Exception e) {
        fail("No se esperaba excepción: " + e.getMessage());
    }
}
```

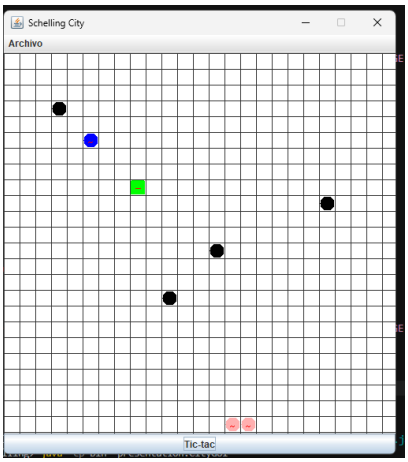
5. Realicen una prueba de aceptación de este par de métodos: iniciando la aplicación exportando a oneCity.txt. saliendo, entrando, creando una nueva e importando el archivo otherCity.txt. ¿Qué resultado obtuvieron? Capturen la pantalla final.



Se abre el archivo de forma correcta y se permite interactuar con la ciudad de forma fluida.

6. 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.

```
Hole 9 20
Hole 12 13
Hole 15 10
Solitaria 23 14
Solitaria 23 15
Solitaria 24 14
Solitaria 24 15
Person 5 5
Walker 8 8
Hole 3 3
```

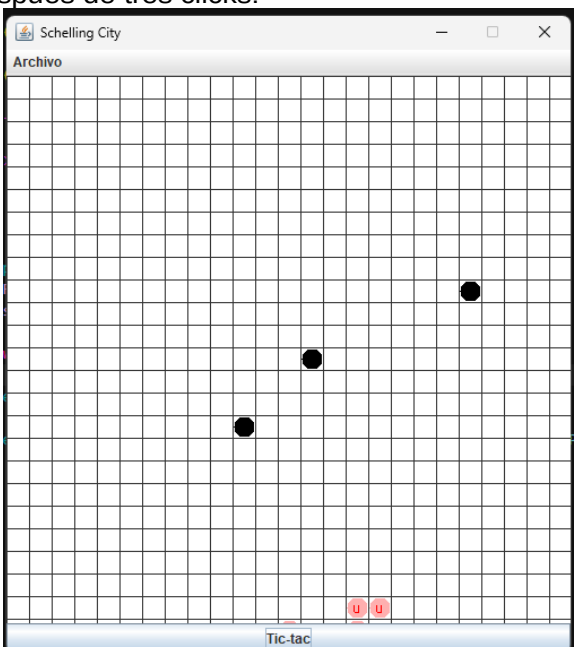


Nota: se hizo un cambio en el formato de la exportación en el siguiente ciclo.

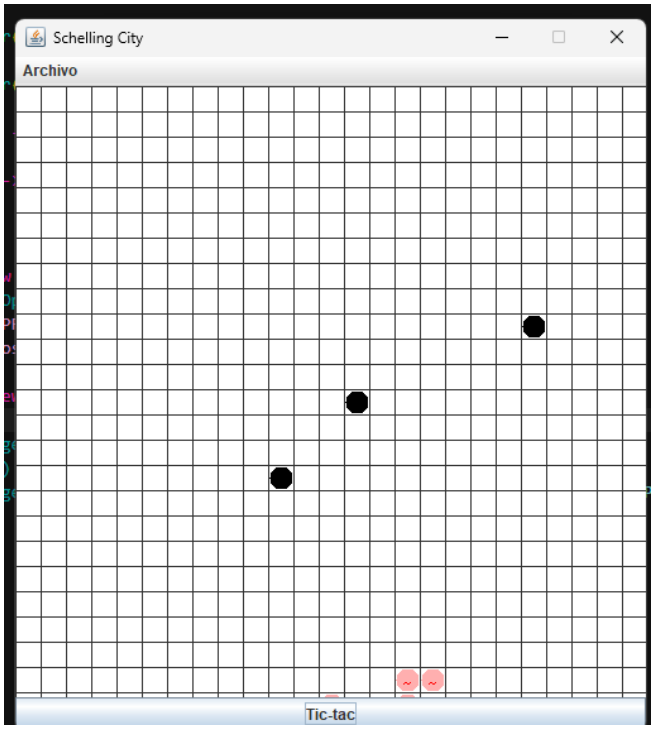
Analizando comportamiento

[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.
   Se guarda el archivo después de tres clicks:

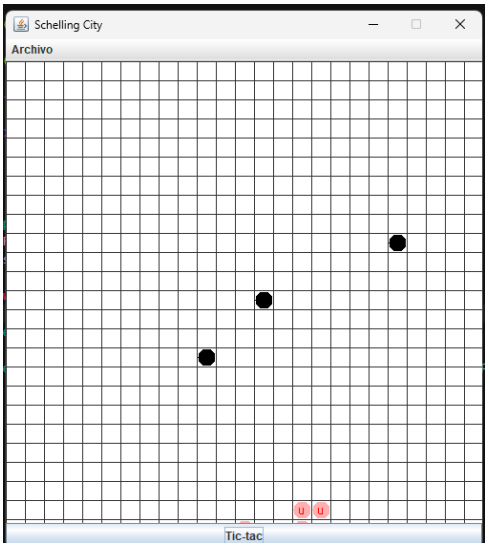


   Al abrir archivo aparece así:

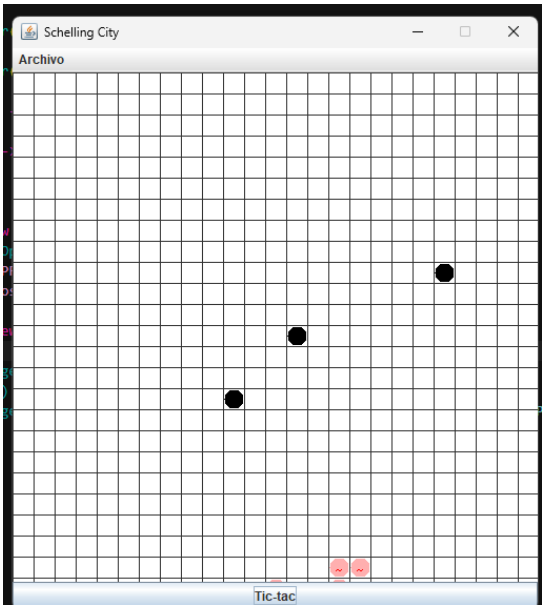


Esto aparece debido a que se toma el estado por defecto de las personas hasta que se hace un movimiento (tic-tac) en el que las personas revisen en que estado están

2. Ejecuten la aplicación, tres clics, exporten a un archivo cualquiera e importen. Describan el comportamiento.
   Exportamos esta ciudad



Nuevamente nos encontramos con el mismo problema:



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

   Son los mismos resultados.

Perfeccionando salvar y abrir
[En lab06.doc, *.asta y *.java] [NO OLVIDEN BDD y MDD]

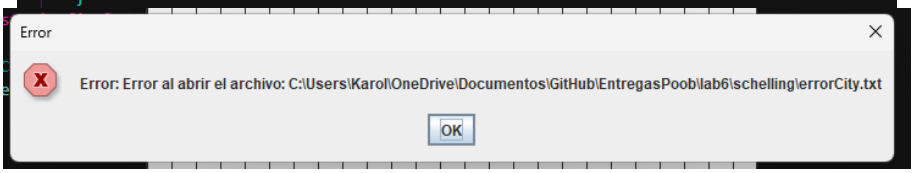
1. Copien las versiones actuales de open y save y renómbrenlos como open01 y save01
2. Perfeccionen el manejo de excepciones de los métodos open y save detallando los errores. No olviden pruebas de unidad.

```java
@Test
public void shouldThrowExceptionWhenSavingToInvalidPath() {
    City ciudad = new City();
    File archivo = new File(pathname:"/no/existe/city.save");  // Ruta inválida

    CityException thrown = assertThrows(expectedThrowable:CityException.class, () -> {
        ciudad.save(archivo);
    });

    assertTrue(thrown.getMessage().contains(s:"Error al guardar el archivo"));
}

@Test
public void shouldThrowExceptionWhenOpeningNonexistentFile() {
    File archivo = new File(pathname:"archivo_que_no_existe.save");

    CityException thrown = assertThrows(expectedThrowable:CityException.class, () -> {
        City.open(archivo);
    });

    assertTrue(thrown.getMessage().contains(s:"Error al abrir el archivo"));
}
```

(olvide el screenshot de resultados de pruebas aquí)

3. Realicen una prueba de aceptación para validar uno de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.
   Se escribe un archivo txt y desde el menú de cityGUI se intenta abrir

```java
package domain;
public class CityException extends Exception {
    public static final String ERROR_ABRIR = "Error al abrir el archivo: ";
    public static final String ERROR_GUARDAR = "Error al guardar el archivo: ";
    public static final String ERROR_EXPORTAR = "Error al exportar la ciudad: ";
    public static final String ERROR_IMPORTAR = "Error al importar la ciudad: ";
    public static final String FORMATO_INVALIDO = "Formato inválido en línea ";
    public static final String COORDENADAS_INVALIDAS = "Coordenadas inválidas en línea ";
    public static final String TIPO_DESCONOCIDO = "Tipo de ítem desconocido en línea ";

    public CityException(String message) {
        super(message);
    }
```



Perfeccionando importar y exportar.

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

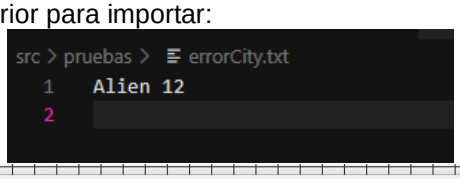
1. Copien las versiones actuales de import y export y renómbrenlos como import01 y export01
2. Perfeccionen el manejo de excepciones de los métodos import y exportdetallando los errores. No olviden pruebas de unidad.

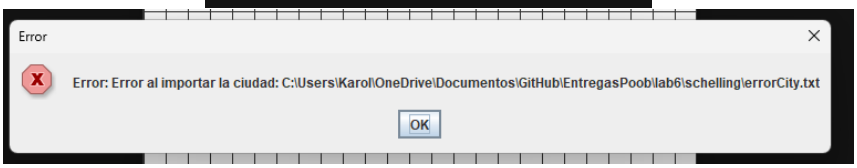
```
.....................
Time: 0,296

OK (21 tests)
```

```java
@Test
public void shouldThrowExceptionWhenInexistentToProtectedPath() {
    City ciudad = new City();
    File archivo = new File(pathname:"/ab/city.txt");

    CityException thrown = assertThrows(expectedThrowable:CityException.class, () -> {
        ciudad.exportar(archivo);
    });

    assertTrue(thrown.getMessage().contains(s:"Error al exportar la ciudad"));
}


@Test
public void shouldThrowExceptionWhenImportingWithInvalidCoordinates() throws IOException {
    File archivo = new File(pathname:"ciudad_mal.txt");
    try (PrintWriter writer = new PrintWriter(archivo)) {
        writer.println(x:"Person a b");  // Coordenadas inválidas
    }

    CityException thrown = assertThrows(expectedThrowable:CityException.class, () -> {
        City.importar(archivo);
    });

    assertTrue(thrown.getMessage().contains(s:"Coordenadas inválidas en línea"));
    archivo.delete();  // Limpieza
}
```

```java
@Test
public void shouldThrowExceptionWhenImportingUnknownItemType() throws IOException {
    File archivo = new File(pathname:"ciudad_tipo_desconocido.txt");
    try (PrintWriter writer = new PrintWriter(archivo)) {
        writer.println(x:"Alien 10 10");  // Tipo inválido
    }

    CityException thrown = assertThrows(expectedThrowable:CityException.class, () -> {
        City.importar(archivo);
    });

    assertTrue(thrown.getMessage().contains(s:"Tipo de ítem desconocido en línea"));
    archivo.delete();  // Limpieza
}
```

3. Realicen una prueba de aceptación para validar uno de los nuevos mensajes diseñados, ejecútenla y capturen la pantalla final.
   Usando el mismo archivo anterior para importar:





Perfeccionando importar. Hacia un minicompilador.

[En lab06.doc, *.asta , cityErr.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, palabra que tiene el error y causa de error.
3. Escriban otro archivo con errores, llámelo cityErr.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 , schellingFlex.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 elementos creados en el futuro. No olviden pruebas de unidad.
   (Investiguen cómo crear un objeto de una clase dado su nombre)
3. Escriban otro archivo de pruebas, llámelo cityErrG.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 ?
   Carolina 12 horas

2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?
   Incompleto debido a falta de tiempo.
3. Considerando las prácticas XP del laboratorio. ¿cuál fue la más útil? ¿por qué?

   NA
4. ¿Cuál consideran fue el mayor logro? ¿Por qué?
   Entender el uso de los métodos de IO para aplicarlos.
5. ¿Cuál consideran que fue el mayor problema técnico? ¿Qué hicieron para resolverlo?

   El mayor problema técnico fue la cuestión de que al guardarse y abrir (o importar y exportar) la ciudad no aparecían de inmediato los estados de animo de las personas, por cuestiones de tiempo no se hizo el arreglo, pero en importar y exportar se consideró guardar el estado de animo de cada persona a la hora de exportarlo para usar un método setState a la hora de importación.

6. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?
   ..
7. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

String (Java Platform SE 8). (2025, 5 abril).

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

PrintWriter (Java Platform SE 8). (2025, 5 abril).
https://docs.oracle.com/javase/8/docs/api/java/io/PrintWriter.html