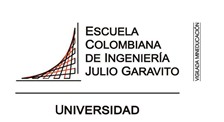
****

**Escuela Colombiana de Ingeniería Julio Garavito**

**Programación Orientada a Objetos 2024-2**

**Laboratorio III**

**Cristian Santiago Pedraza Rodríguez**

**Andersson David Sánchez Méndez**

**4 de octubre de 2024**

# PROGRAMACIÓN ORIENTADA A OBJETOS

**Herencia e interfaces ADEMAS Java desde consola 2024-2**

# Laboratorio 3/6

**OBJETIVOS**

Desarrollar competencias básicas para:

1. Aprovechar los mecanismos de la herencia y el uso de interfaces.
2. Organizar las fuentes en paquetes.
3. Usar la utilidad jar de java para entregar una aplicación.
4. Extender una aplicación cumpliendo especificaciones de diseño, estándares y verificando su corrección.
5. Vivenciar las prácticas XP : Code must be written to agreed [standards](http://www.extremeprogramming.org/rules/standards.html). Code the [unit test first](http://www.extremeprogramming.org/rules/testfirst.html).
6. Utilizar los programas básicos de java (javac, java, javadoc, jar), desde la consola.

# 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

# Contexto

**Replicate** es un nuevo proceso para producción en masa de piezas idénticas. El proceso usa una reja bidimensional de celdas. En las celdas se pueden colocar diferentes cosas.

Una de estas cosas son las células que tienen dos estados: "activa" o "inactiva".

En una secuencia de pasos discretos, cada cosa en la red actualiza simultáneamente su estado al examinar su propio estado y la de sus ocho vecinas circundantes.

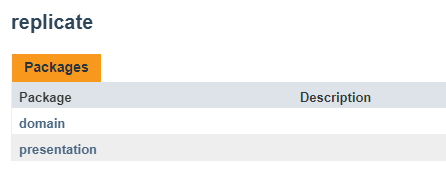
**Conociendo** [En lab03.doc y replicate.asta ]

1. En el directorio descarguen los archivos contenidos en replicate.zip. Revisen el código:

a) ¿Cuántos paquetes tiene?

Tiene dos paquetes



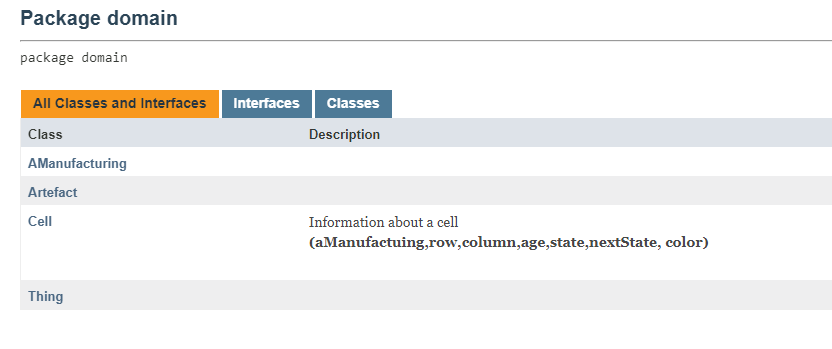


b) ¿Cuál es el propósito del paquete presentación?



El propósito de este paquete es agrupar todas las clases que tienen como misión y trabajo, hacer la representación gráfica de la interfaz del usuario. (GUI).

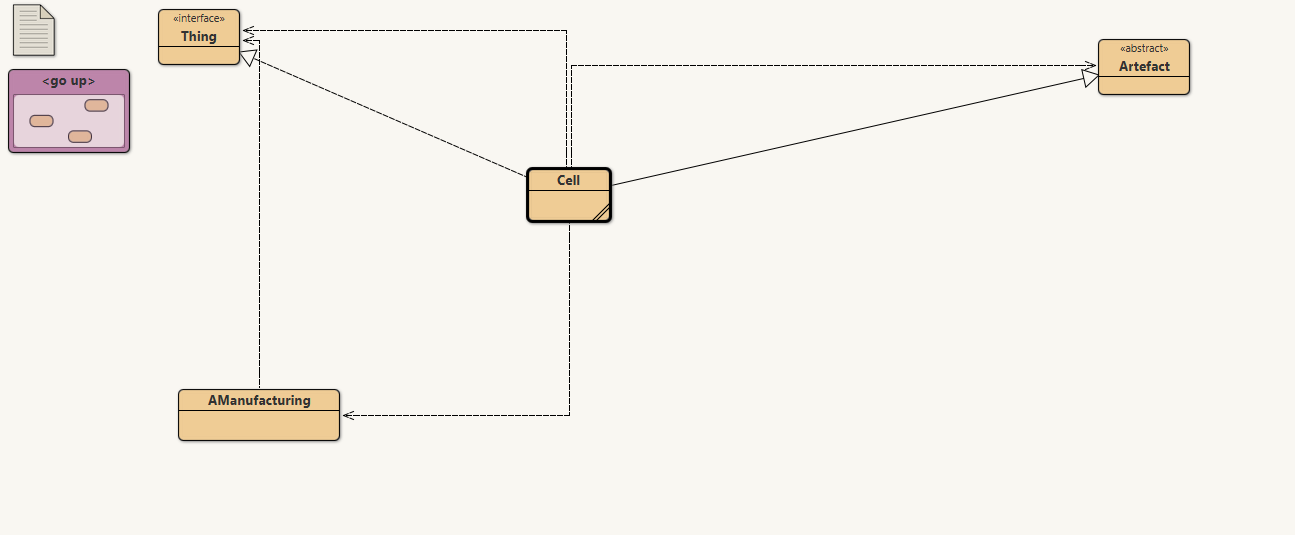
c) ¿Cuál es el propósito del paquete dominio?



El paquete de dominio se encarga de agrupar todas las clases, interfaces y todos los componentes software en general que trabajan en la resolución del dominio del problema.

1. Revisen el paquete de dominio,

a)¿Cuáles son los diferentes tipos de componentes de este paquete?



En el paquete domain están las clases concretas, la clase abstracta y la interfaz

b) ¿Qué implica cada uno de estos tipos de componentes?

Clases concretas: Clase en la que sí se pueden crear objetos

Clases abstractas: Clase en la que no se puede definir o construir instancias(objetos)

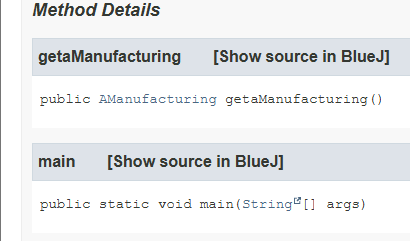
Interfaz: Colección de métodos y propiedades constantes.

1. Revisen el paquete de presentación,

a) ¿Cuántos componentes tiene?

En el paquete de dominio solo hay un componente. El componente en cuestión es una clase concreta.

b) ¿Cuántos métodos públicos propios (no heredados) ofrece? 2



1. Para ejecutar un programa en java,

¿Qué método se debe ejecutar? main()

¿En qué clase se encuentra? En el paquete presentation, en la clase principal AManufacturingGUI

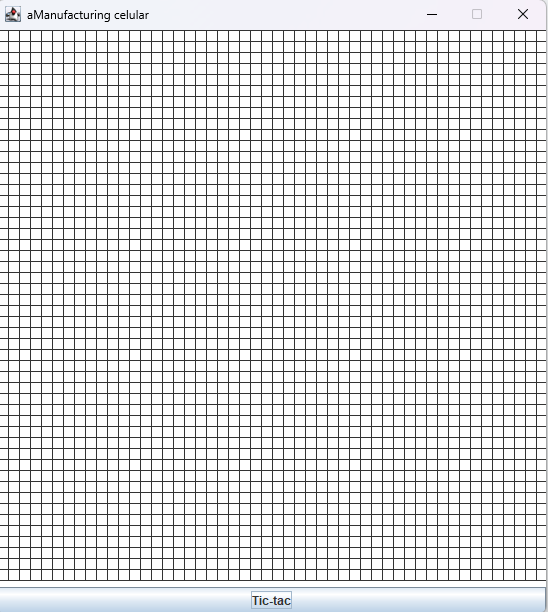
1. Ejecuten el programa.

¿Qué funcionalidades ofrece? Ofrece la funcionalidad de Tic -tac

¿Qué hace actualmente? Actualmente el programa no hace nada.

¿Por qué? Esto se da ya que el cuerpo del método Tic-tac está vacío.

# (Deben ejecutar la aplicación java, no crear un objeto como lo veníamos haciendo)



**Arquitectura general.** [En lab03.doc y replicate.asta]

1. Consulte el significado de las palabras package e import de java.

**PACKAGE**

¿Qué es un paquete?

Mecanismo que se utiliza para organizar las clases e interfaces en módulos o grupos. Funciona como un "espacio de nombres" que ayuda a evitar conflictos entre clases con el mismo nombre en diferentes paquetes.

**Sintaxis**: Se define al principio del archivo fuente.  
package com.miproyecto.utilidades;

¿Para qué sirve?

* **Agrupación de clases**: Permiten agrupar clases relacionadas de manera lógica.
* **Estructura de nombres**: Previenen colisiones de nombres de clases.
* **Accesibilidad**: Ayudan a definir la visibilidad de las clases y los métodos. Si una clase no es pública, solo será accesible dentro de su propio paquete.

**IMPORT**

¿Para qué se usa?

Se utiliza para importar clases o paquetes en un archivo Java, lo que permite acceder a esas clases o interfaces sin necesidad de escribir el nombre completo de su paquete cada vez que que se quiera usar.

**Sintaxis**:  
import nombre\_del\_paquete.NombreDeLaClase; // o \* para llamar a todos

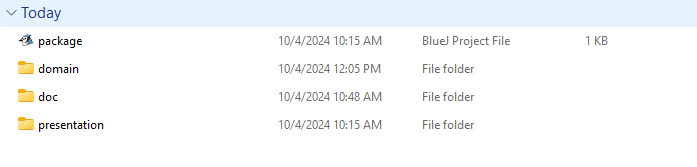
¿Para qué se importa? Explique su uso en este programa.

* **Facilita el uso de clases**: En lugar de referirse a las clases con su nombre completamente calificado (como java.util.Scanner), se usa el import para hacerlo más legible.
* **Reutilización de código**: Si el programa necesita utilizar una clase o un conjunto de clases que están definidas en otro paquete, import permite acceder a ellas de forma sencilla.

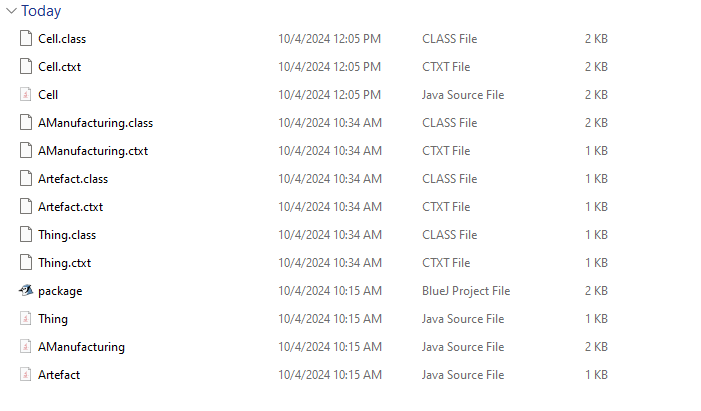
En Java, las palabras clave **package** e **import** están relacionadas con la organización y el uso del código en módulos más pequeños y reutilizables.

1. Revise el contenido del directorio de trabajo y sus subdirectorios. Describa su contenido. ¿Qué coincidencia hay entre paquetes y directorios?

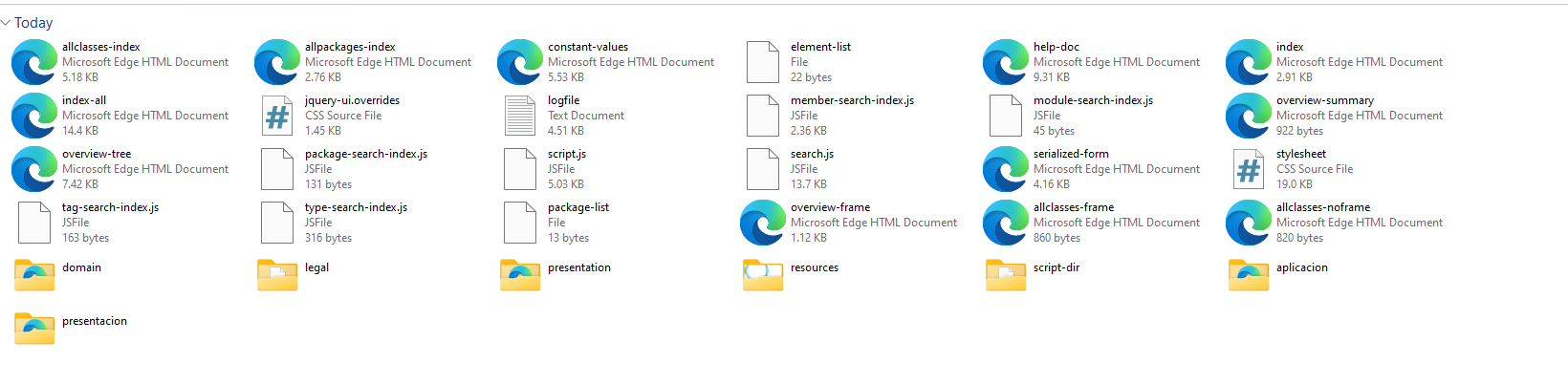




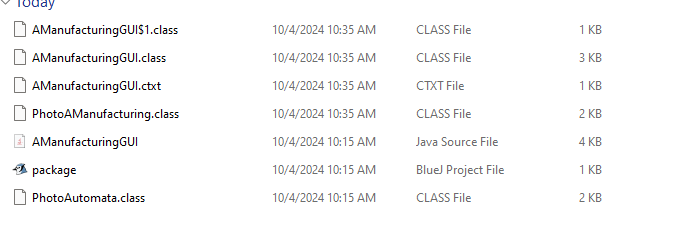
En domain



En doc



En presentation



¿Qué coincidencia hay entre paquetes y directorios?

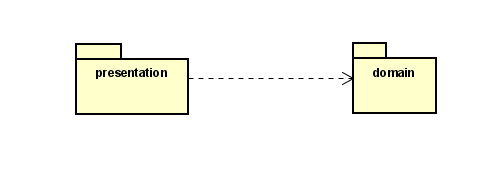
En Java, los **paquetes** y los **directorios** tienen una coincidencia: ambos organizan y estructuran el código.

* **Paquetes**: Agrupar clases relacionadas. Proporcionar una estructura jerárquica. Por ejemplo, el paquete com.example.domain indicaría que la clase pertenece al paquete domain dentro de example y a su vez dentro de com.
* **Directorios**: En el sistema de archivos, cada paquete en Java se refleja como un directorio. Con el ejemplo de paquete com.example.domain, cada clase en ese paquete estará representada por un archivo .class o .java dentro de esa carpeta.

Ahora, con base a lo anterior, podemos afirmar que la coincidencia radica en que cada directorio de trabajo tiene asociado un paquete asociado.

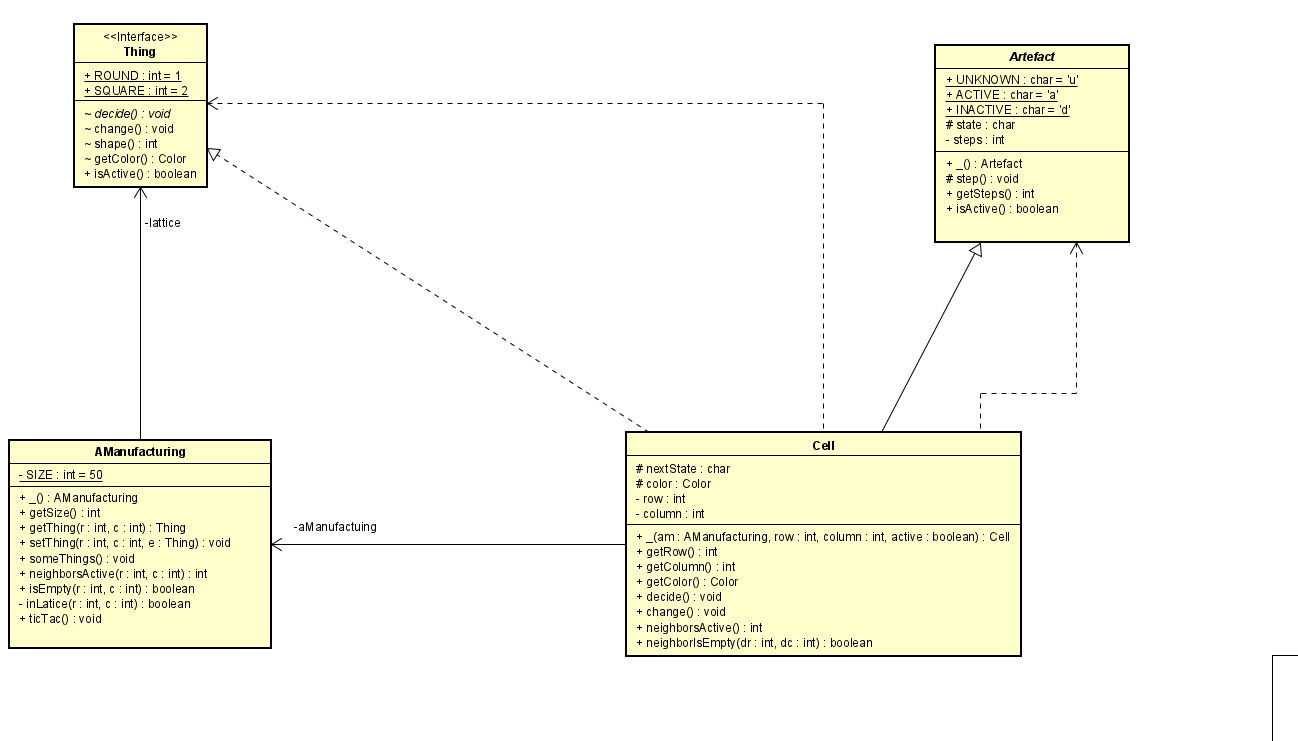
1. Adicione al diseño la arquitectura general con un diagrama de paquetes en el que se presente los paquetes y las relaciones entre ellos. Consulte la referencia en moodle. **En astah, crear un diagrama de clases (cambiar el nombre por Package Diagram0)**

****

****

**Arquitectura detallada.** [En lab03.doc y replicateasta]

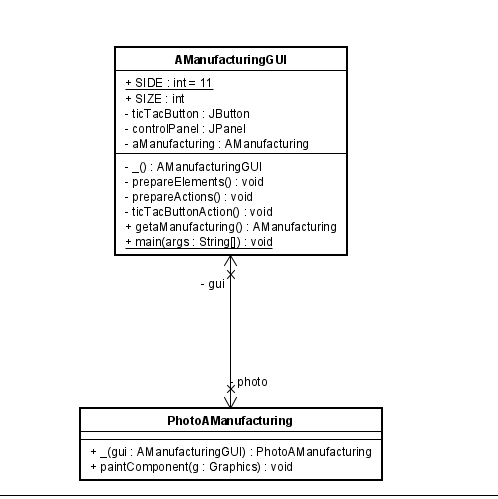
1. Para preparar el proyecto para **BDD.** Completen el diseño detallado del paquete de dominio. Adicionen el diagrama de clases en el paquete correspondiente.

****

a) ¿Qué componentes hacían falta?

Hacían falta relaciones, algunos atributos y métodos y algunas clases no estaban creadas como Cell, y la interfaz Thing.

1. Realicen el diseño detallado del paquete de presentación. Adicionen el diagrama de clases al paquete correspondiente.



a) ¿Por qué hay dos clases y un archivo .java?

Solo se ha utilizado un archivo .java porque ambas clases están encapsuladas dentro de la clase principal AManufacturingGUI. No están separadas en archivos distintos, ya que la clase PhotoAManufacturing es una clase interna que complementa y trabaja en conjunto con AManufacturingGUI.

¿Por qué?

La razón detrás de tener ambas clases en un solo archivo es su relación directa dentro de la interfaz gráfica de usuario. AManufacturingGUI extiende JFrame y maneja la ventana principal de la aplicación, mientras que PhotoAManufacturing extiende JPanel y se encarga de representar los elementos gráficos en el panel.

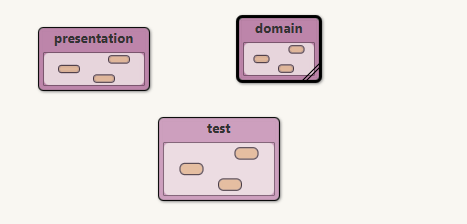
1. Adicione la clase de pruebas unitarias necesaria para **BDD** en un paquete independiente de test**.** (No lo adicione al diagrama de clases)

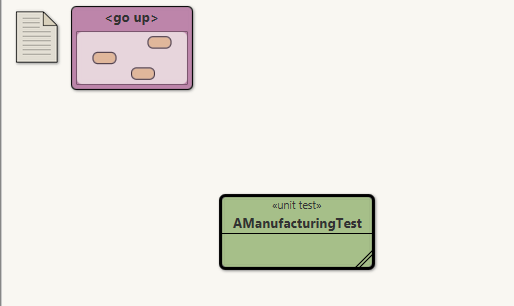
**¿Qué paquete debe usar?** El paquete domain.

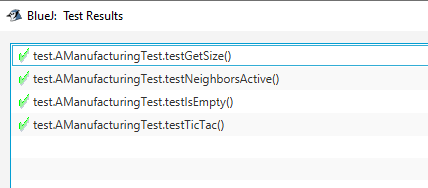
**¿Por qué? ¿Asociado a qué clase?**

Porque es el paquete que tiene toda la estructura del programa, presentation solo tiene la parte de GUI(visualización). Se asocia a la clase principal AManuFacturing.

**¿Por qué?** Ya que es la que tiene el método principal de este laboratorio ticTac() y otros métodos importantes como NeighborsActive.







**Ciclo 1. Iniciando con las células normales [En lab03.doc y \*.java]**

# (NO OLVIDE BDD – MDD)

1. Estudie la clase AManufacturing

**¿Qué tipo de colección usa para albergar cosas?**

Usa una matriz bidimensional de Thing (Thing[][] lattice) para almacenar y organizar los objetos que representan cosas en su "rejilla" (lattice), de acuerdo a coordenadas específicas.

**¿Puede recibir células? ¿Por qué?**

Sí, AManufacturing puede recibir objetos de tipo Cell, ya que esta clase(Cell) implementa la interfaz Thing, lo cual le permite ser parte de la matriz lattice. El método setThing(int r, int c, Thing e) permite asignar cualquier objeto que implemente Thing en posiciones específicas dentro de esta matriz bidimensional.

1. Estudie el código asociado a la clase Cell,

**¿en qué estado se crea?**

Se crea en estado activo o inactivo dependiendo del parámetro active en el constructor. Está controlado por: state=(active ? Artefact.ACTIVE: Artefact.INACTIVE);

**¿qué forma usa para pintarse?**

La implementación de Thing al ser que Cell implementa Thing indica que la forma por defecto es SQUARE. Se podría sobrescribir este método en Cell.

**¿cuándo aumenta su tiempo?**

Aumenta su tiempo (steps) cada vez que se llama al método change(), el cual incrementa el contador de pasos llamando al método step() de su clase abstracta Artefact(steps++).

**¿qué clases definen la clase Cell ?**

Artefact: hereda el estado (state) y el método step() para contar los pasos.

Thing: define la forma, el color, y la capacidad de decidir y cambiar su estado.

1. Cell por ser un Artefact,

**¿atributos tiene?**

nextState, color, y posiciones row y column.

**¿qué puede hacer (métodos)?**

decide()), change(), getColor(), y neighborsActive().

**¿qué decide hacer distinto?**

Cell toma decisiones dependiendo alpróximo estado en función del método decide().

**¿qué no puede hacer distinto a todos los artefactos?**

Cell no puede cambiar su estado directamente, porque por esto existe el método change() para actualizar su estado con el método ticTac().

**¿qué debe aprender a hacer? Justifique sus respuestas.**

Cell debería estar en la capacidad de interactuar con otras células, tomar decisiones en base al estado de sus vecinos(activo, inactivo), y aprender a modificar su color. Debe aprender a hacer eso para no estar dependiendo de otras clases para el uso de esos métodos.

1. Por comportarse como un Thing,

**¿qué sabe hacer?**

Cell se comporta como un Thing en el sentido de que puede ser pintado en una forma definida (SQUARE), tiene un color asociado(red), y un método isActive() que verifica su estado.

**¿qué decide hacer distinto?**

Cell sobreescribe métodos de la interfaz Thing para definir su propia forma y comportamiento, como decidir cuándo cambia de color o forma.

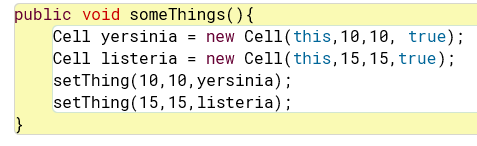
**¿qué no puede hacer distinto?**

La regla está en que Cell implementa la interfaz Thing, así que siempre estará ligada, a menos que no esté implementada como interfaz; la forma base y método change() siguen lo que haga Thing.

**¿qué debe aprender a hacer? Justifique sus respuestas.**

Cell debería ser más independiente de algunos métodos como es lo es change() en donde no dependa directamente de lo que hace Thing, es decir, teniendo ese método abstracto y no como default en la interfaz Thing, cada clase que la implemente puede usar el método pero de manera diferente.

1. Ahora vamos a crear dos células en diferentes posiciones (10,10) (15,15) llámelas yersinia y listeria usando el método someThings() . Ejecuten el programa,

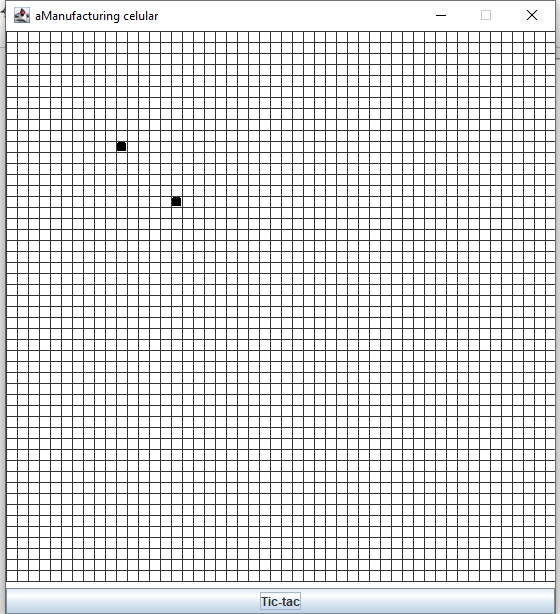


**¿Qué pasa con las células?**

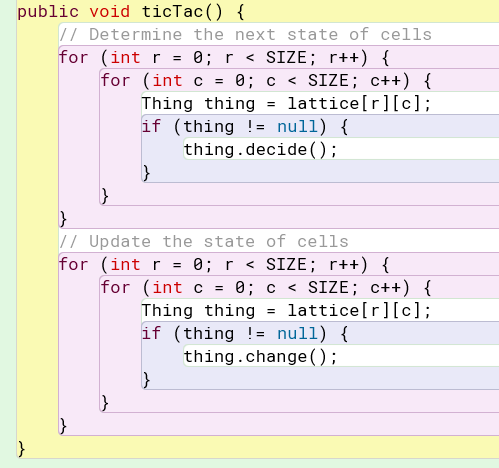
**¿Por qué?**

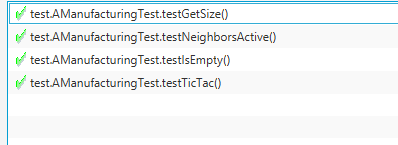
Vemos que las células que se crearon se ubicaron perfectamente en la posición (10,10) y (15,15) respectivamente, esto pasa porque en el constructor de AManufacturingGUI se obtiene el tamaño de AManufacturing que es donde se está referenciando con el método someThings() para tener presente el tamaño del grid, y cambiandoles de estado a activo, con true; y con el constructor de AManufacturing se ubican las células a la posición establecida.

Capturen una pantalla significativa.



1. Diseñen, construyan y prueben el método llamado ticTac() de la clase AManufacturing .





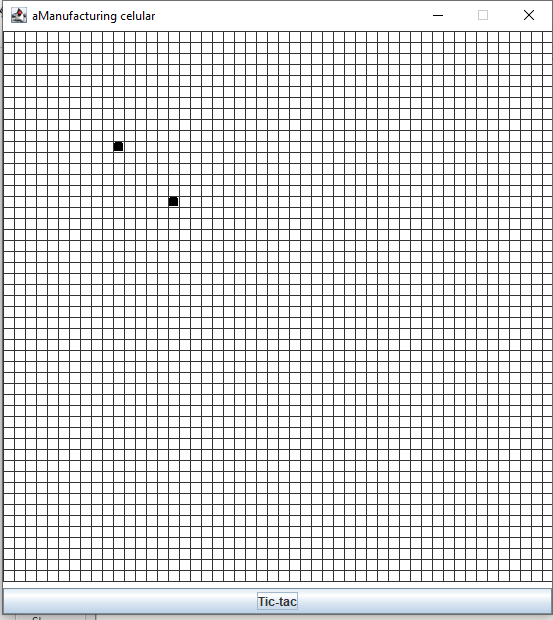
1. ¿Cómo quedarían yersinia y listeria después de uno, dos, cuatro y seis **Tic-tac**?

* Ejecuten el programa. Capturen pantallas significativas en momentos correspondientes.

¿Es correcto? Sí

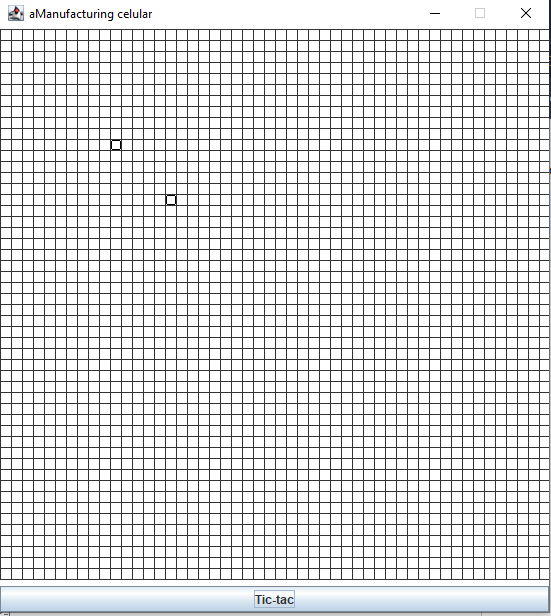
**FIRST MOMENT**

Ambas células (yersinia y listeria) se encuentran en estado activo porque comenzaron activas y, dado que el primer tic-tac es un paso impar, su estado no cambia, así que ambas células permanecen visibles.



**SECOND MOMENT**

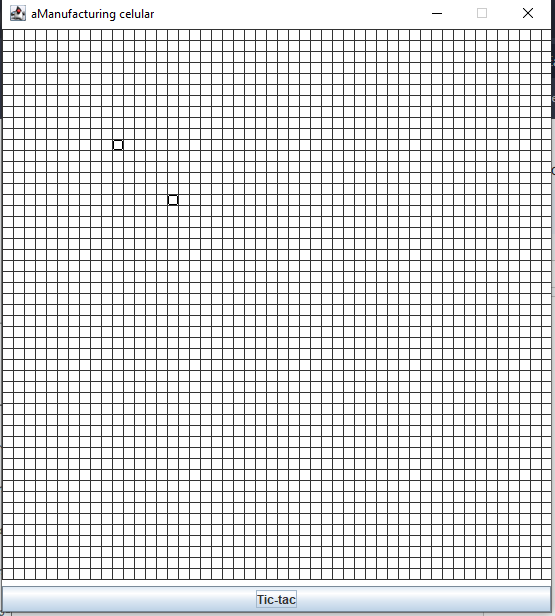
En el segundo tic-tac, el método decide() hace que ambas células cambien a estado inactivo debido a la lógica de alternancia (paso par), así que ambas células pasan a estado invisible.



**FOURTH MOMENT**

En el tercer tic-tac, ambas células vuelven a estar activas, así que se hacen visibles nuevamente.

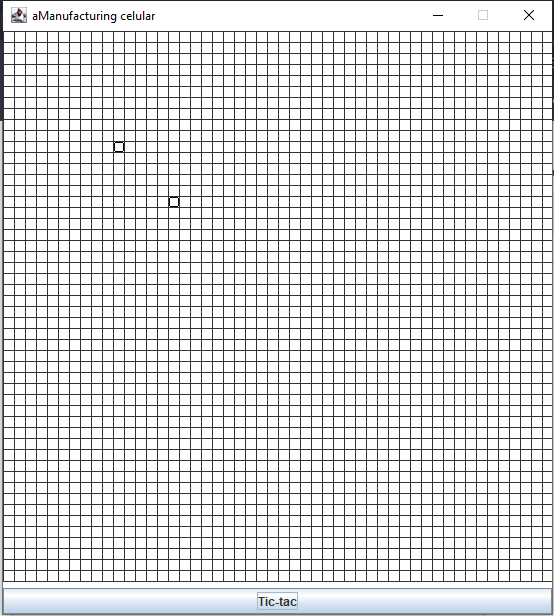
Al llegar al cuarto tic-tac (otro paso par), ambas células cambian nuevamente a estado inactivo y se hacen invisibles otra vez.

****

**SIXTH MOMENT**

El comportamiento de alternancia se mantiene. Así que, después del quinto tic-tac, las células estarán visibles nuevamente.

Luego, en el sexto tic-tac, vuelven a cambiar a estado inactivo y se hacen invisibles.



**Ciclo 2. Incluyendo a las células turistas** [En lab03.doc y replicateasta]

# (NO OLVIDE BDD – MDD)

El objetivo de este punto es permitir recibir células turistas. Ellas (i) son de color naranja activas y amarillo inactivas, inician activas; (ii) si están activas, tratan de moverse hacia el centro de la reja (iii) si están inactivas, tratan de moverse hacia los bordes de la reja (iv) cambian de estado, si no se pueden mover.

1. Para implementar esta nueva célula
   * Cabe aclarar que para la implementación de estas células turistas, se extendió la clase Cell



**¿cuáles métodos se sobre-escriben (overriding)?**

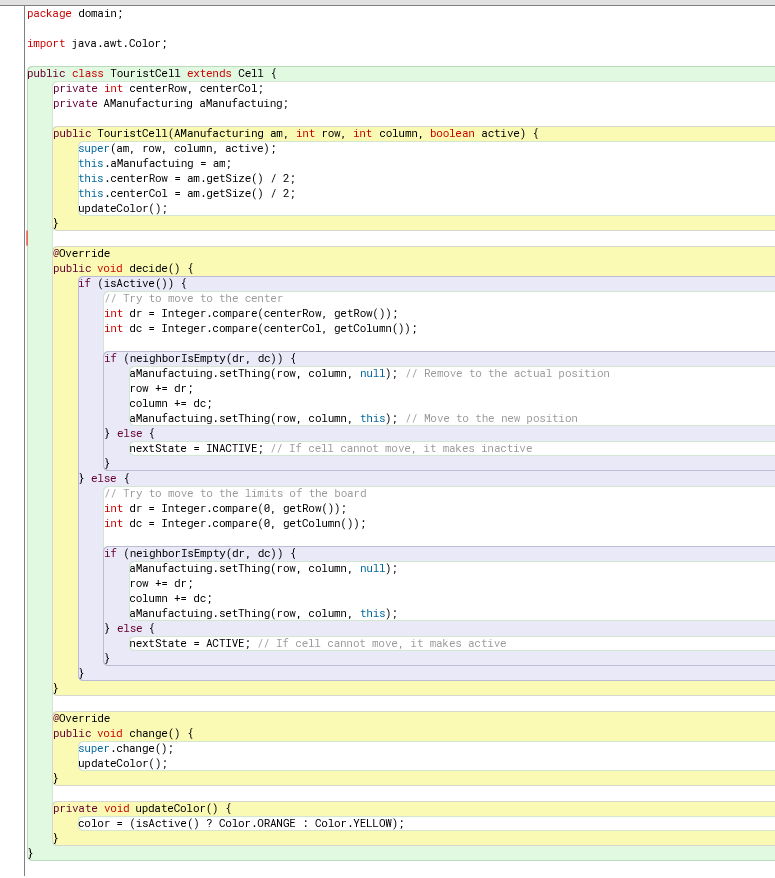
Se sobreescriben los métodos decide() y change(). Decide() porque dependiendo del estado que tenga la célula, se decide hacia dónde se va a desplazar, si para el centro o para el borde; y en change() que se sobreescribe para actualizar el estado de la célula si no se puede mover.

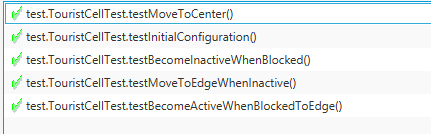
**¿cuáles nuevos atributos se requieren?**

En la nueva clase se requieren tres nuevos atributos y estos son:



1. Diseñen, construyan y prueben esta nueva clase.

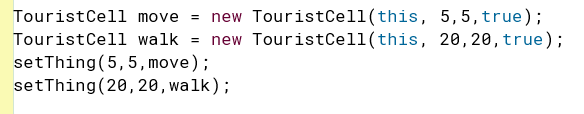


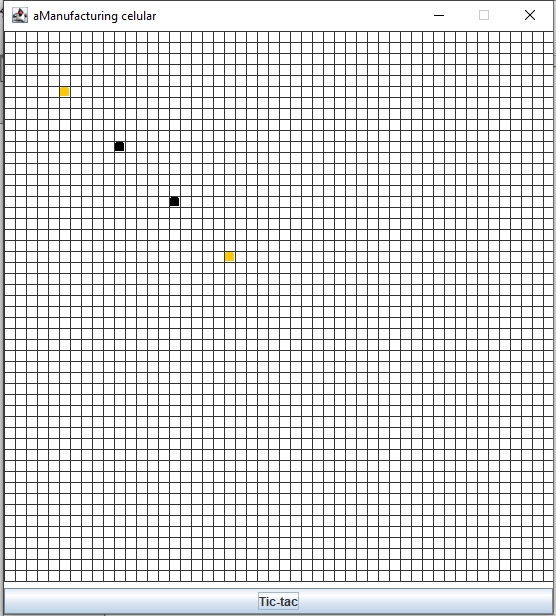


1. Adicione una pareja de turistas, llámelas move y walk,

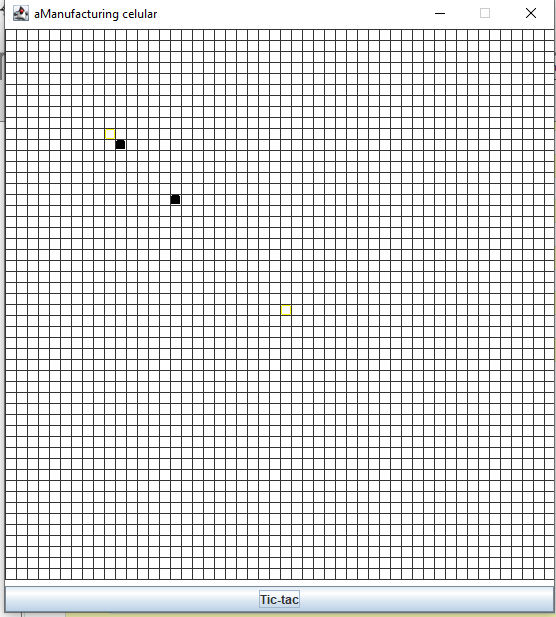
(a) ¿Cómo quedarían después de tres **Tic-tac**? Ejecuten el programa y hagan tres clics en el botón. Capturen una pantalla significativa.

INITIAL CONFIGURATION

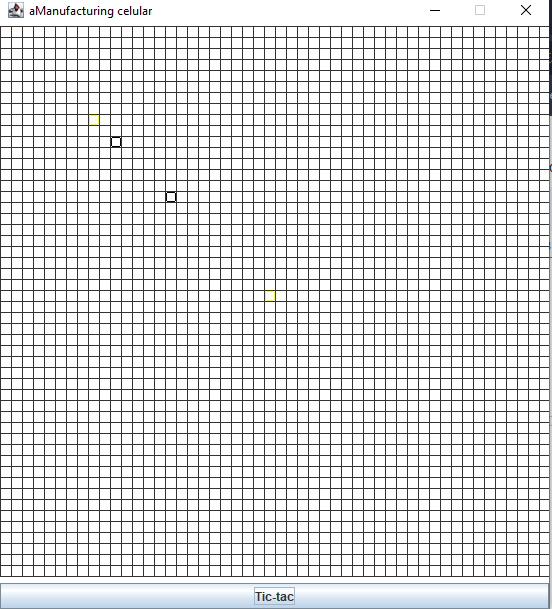




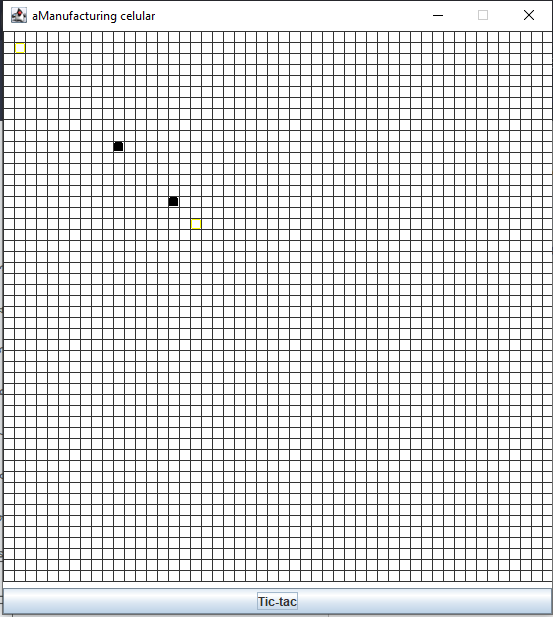
**FIRST MOMENT**

****

**SECOND MOMENT**

****

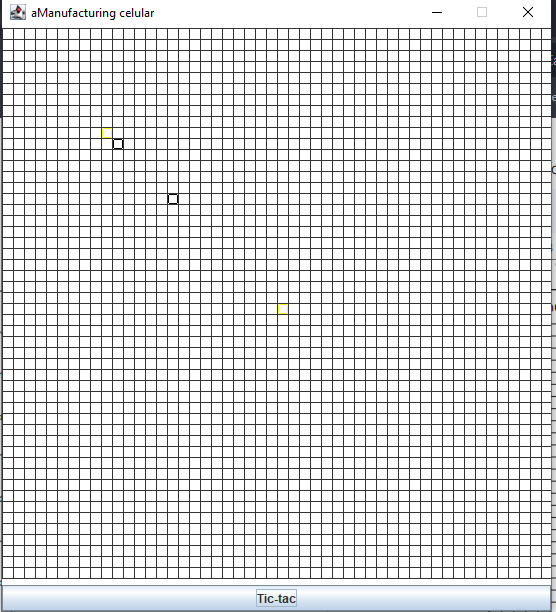
**THIRD MOMENT**

****

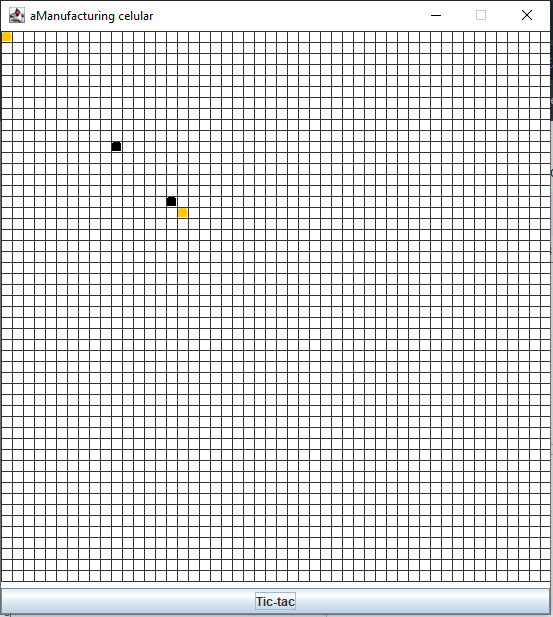
(b) ¿Es correcto?

Sí porque inicia move y walk con estado activa(color naranja, luego cuando se ejecuta ticTac como el estado es activo, entonces intentan moverse hacia el centro del grid, una vez se cumpla eso, como una célula ya está en el centro, pero la otra no puede estar al mismo tiempo, entonces ya no hay más movimientos, y las células pasan de estado activo a inactivo, cambiando a color amarillo, y ahora como tienen ese color, entonces intentar ir al borde del grid, y cuando una ya haya llegado, la otra no podrá moverse, entonces las células cambian de estado otra vez.

**FIRST CASE**



**SECOND CASE**



**Ciclo 3. Adicionando veneno** [En lab03.doc, replicate.asta y \*.java]

El objetivo de este punto es incluir en el replicate veneno en las celdas (sólo vamos a permitir un tipo de veneno). El veneno es cuadrado e inicia en uno de los colores del arco iris (al azar) y va pasando en orden por ellos en cada tic-tac.

# (NO OLVIDE BDD – MDD)

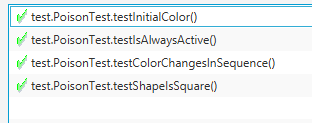
1. Construyan la clase Poison para poder adicionarla en el replicate

¿Debe cambiar en el código del ticTac en algo? ¿por qué?

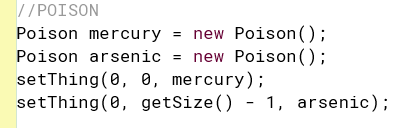
La clase AManufacturing como ya tiene el método ticTac() que llama a decide() y change() para cada objeto Thing en la rejilla. No fue necesario hacer cambios en ticTac(), ya que el Poison puede implementar su propio comportamiento de cambio de color dentro del método change() (@Override), que será invocado por ticTac().

1. Diseñen ,construyan y prueben esta nueva clase.



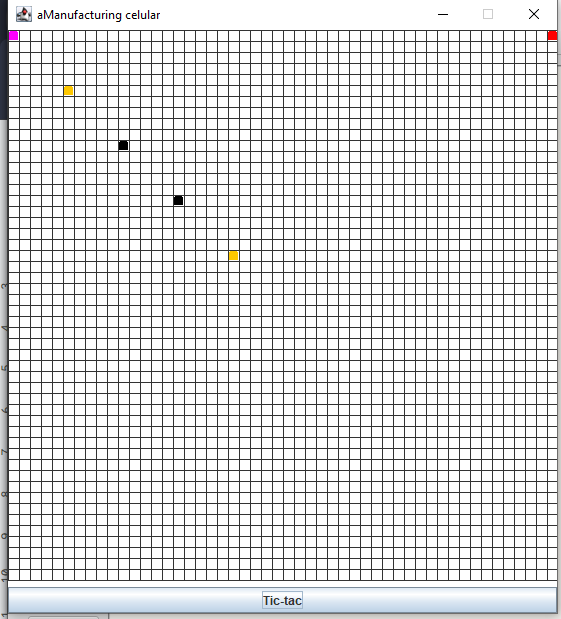


1. Adicionen gotas de veneno en la esquina superior izquierda y superior de replicate, llámenlos mercury y arsenic,

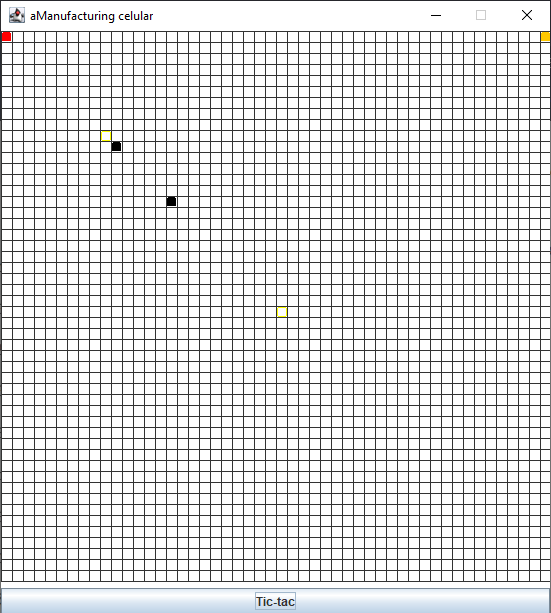


(a) ¿Cómo quedarían después de cuatro **Tic-tac**? Ejecuten el programa y hagan cuatro clics en el botón. Capturen una pantalla significativa.

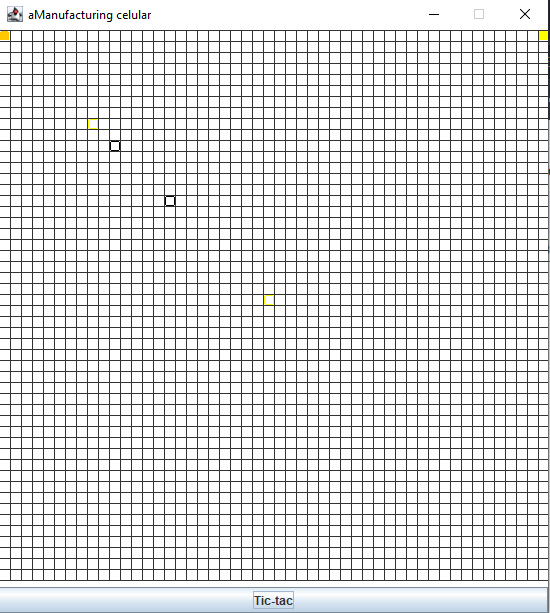
**INITIAL CONFIGURATION**

****

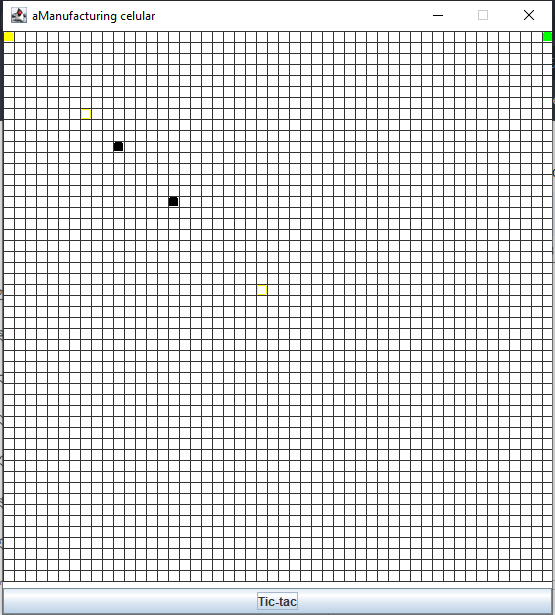
**FIRST MOMENT**

****

**SECOND MOMENT**

****

**THIRD MOMENT**

****

**FOURTH MOMENT**



(b) ¿Es correcto?

Si, vemos que en la configuración inicial se agrega veneno a las dos esquinas superiores con un color aleatorio, y una vez se haga ticTac, las células se mueven correctamente dependiendo en el estado que tenga, ya sea naranja o amarillo, también se verificó que si al momento de que una célula tenga estado inactivo y quiera moverse hacia un borde, pero ya hay veneno, entonces la regla dice que quedaría atascada(no se puede mover más), así cambia su estado a activo(color naranja). Cumple con todas las validaciones del veneno y con las respectivas pruebas.

# Ciclo 4. Nueva célula: Proponiendo y diseñando

El objetivo de este punto es permitir recibir en un nuevo tipo de célula.

# (NO OLVIDE BDD – MDD)

1. Propongan, describan e Implementen el nuevo tipo de células. (Mínimo dos pruebas de unidad)

Propondremos una célula llamada ReflectiveCell. Las características de esta célula serán:

1. **Movimiento**:

* Cuando la célula está activa, intenta moverse de forma aleatoria en una de las cuatro direcciones cardinales (arriba, abajo, izquierda, derecha).
* Si encuentra un obstáculo o un borde, "rebota" y cambia su dirección en la dirección opuesta en el próximo tic-tac.

1. **Estado**:

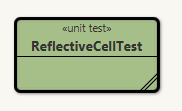
* Si la célula intenta moverse y encuentra un Poison, se volverá inactiva.
* Si está inactiva y encuentra un Poison en su vecindad en el próximo tic-tac, se vuelve activa nuevamente.

1. **Color**:

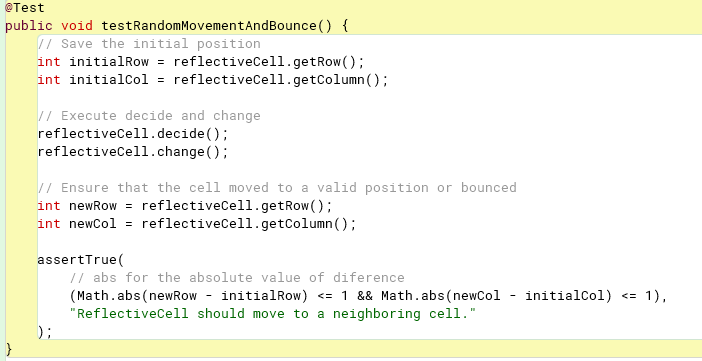
* Cuando está activa, el color de la célula es verde.
* Cuando está inactiva, el color de la célula es gris.

Para hacer la implementación, el modificador de acceso de inLatice es protected.

**Pruebas de unidad**

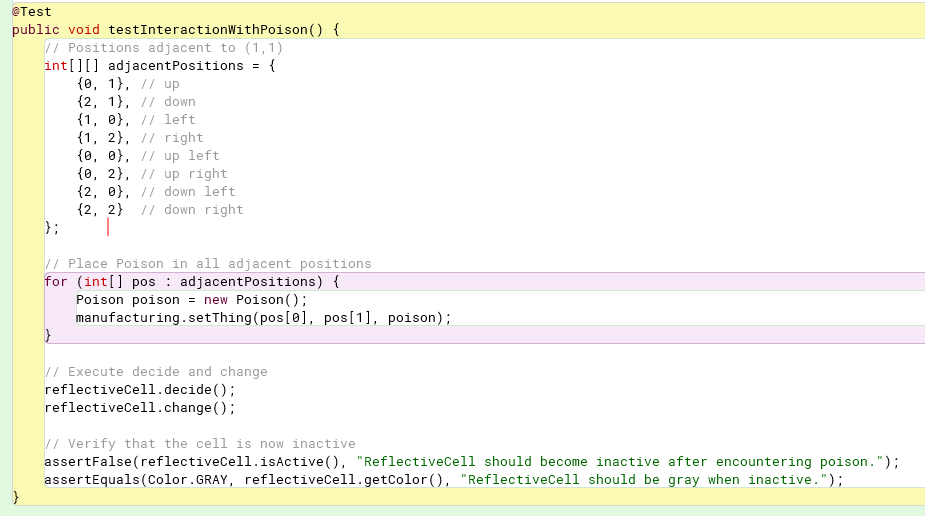
****

1. Al hacer tic-tac, la célula reflectiva se mueve a la celda contigua de forma correcta.

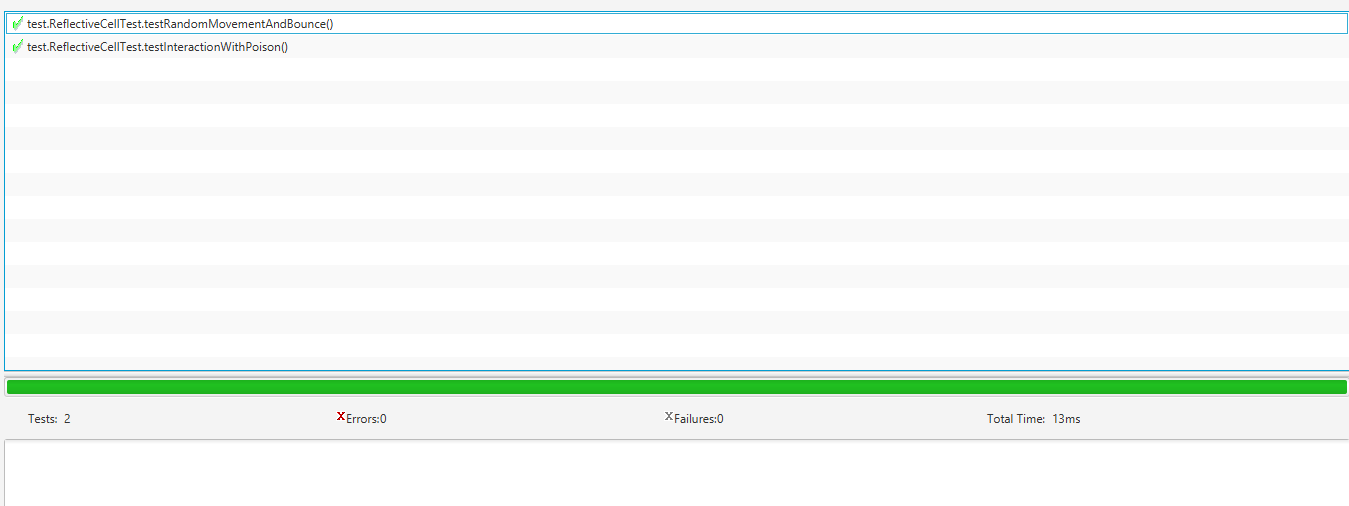


1. Considerando una celda reflexiva, y teniendo su naturaleza aleatoria (se puede mover hacia cualquier dirección dependiendo si está activa). Queremos verificar que si encuentra una celda con poison, se volvera inactiva. Para esto, colocaremos poison en cada una de las celdas adyacentes a la reflectiveCell

Al encontrar el poison, su color debe ser gray



**Compilación de las pruebas de unidad**

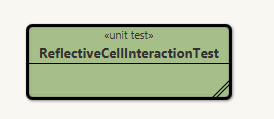
****

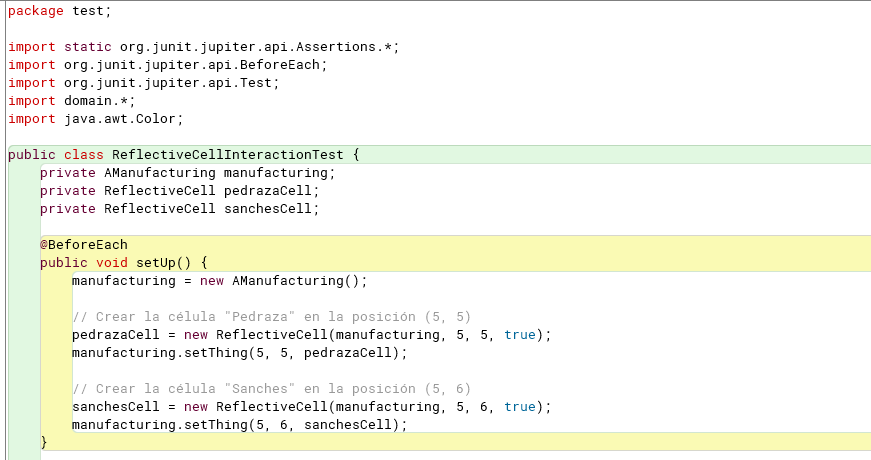
1. Considerando una pareja de ellas con el nombre de ustedes.

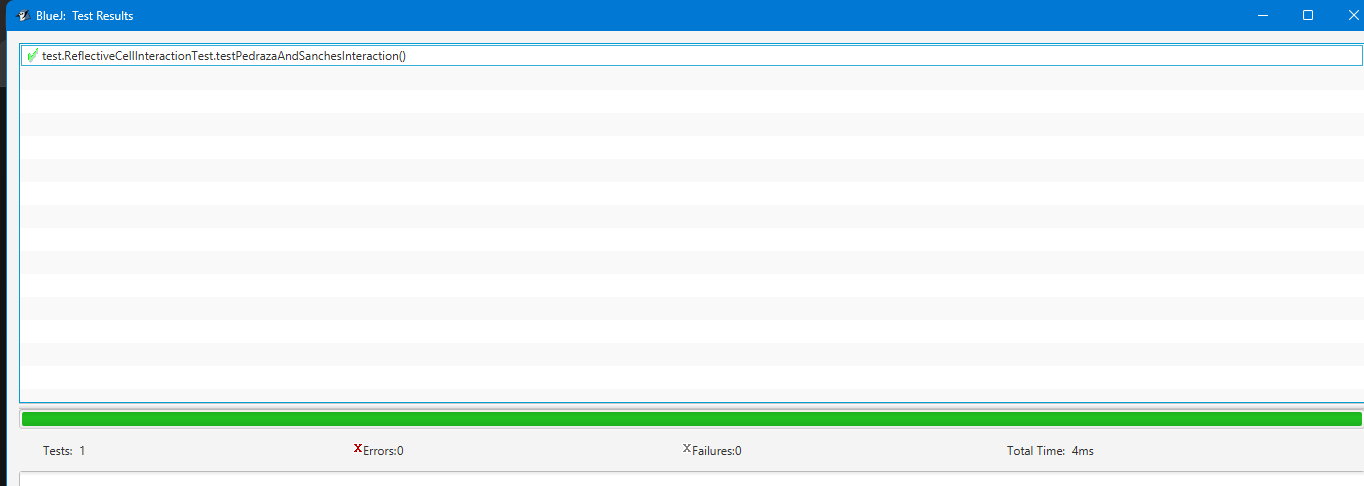
(a) Piensen una prueba significativa y expliquen la intención.

La prueba verifica que dos `ReflectiveCell` ("Pedraza" y "Sanchez") se mueven correctamente dentro de los límites de la rejilla y cambian de dirección al encontrarse, asegurando que no se superpongan y manejen correctamente sus interacciones. Además, se prueba que reaccionan al `Poison` cambiando de estado, evaluando su comportamiento en un entorno dinámico.

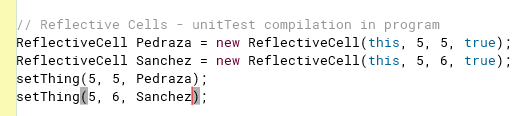
(b) Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados.



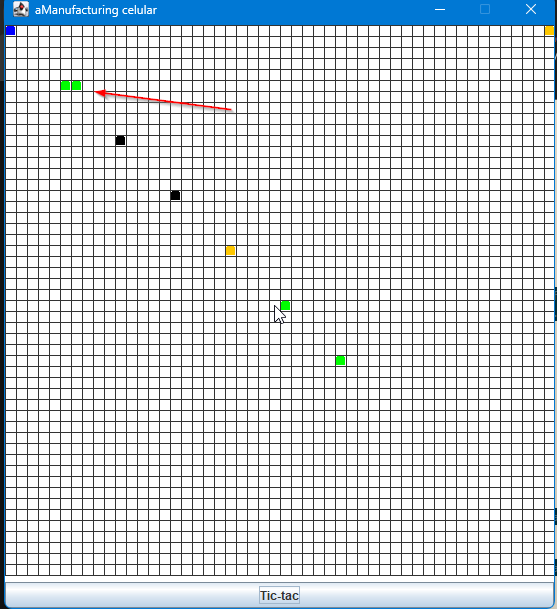




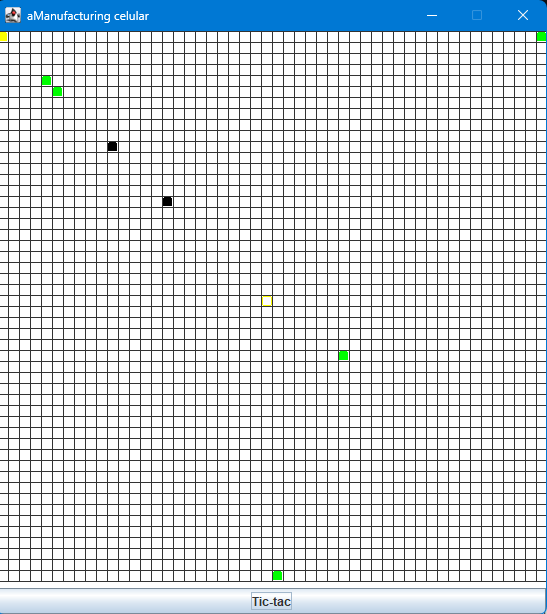
(c) Ejecuten el programa con esa prueba para capturar las pantallas correspondientes.



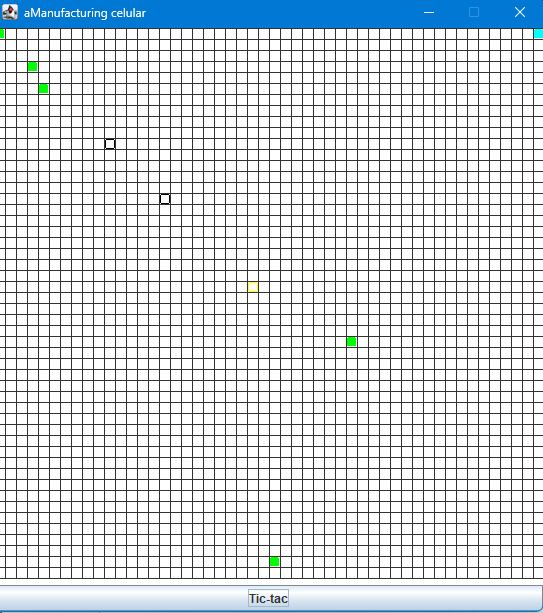
**INITIAL CONFIGURATION**



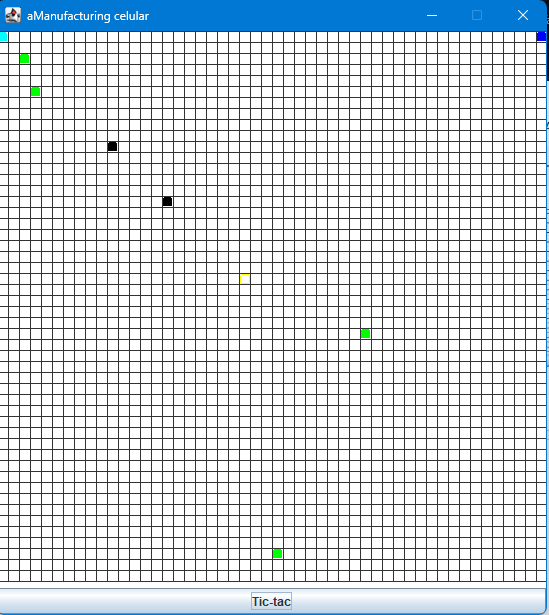
**FIRST MOMENT**

****

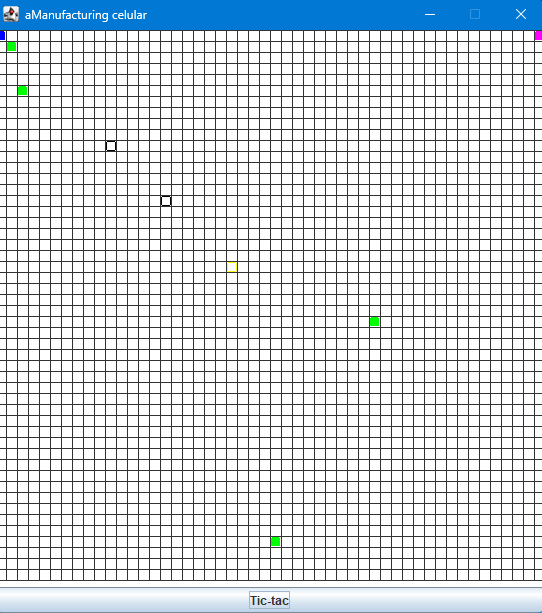
**SECOND MOMENT**

****

**FOURTH MOMENT**

****

**FIFTH MOMENT**

****

Resultado, ambas están dentro de los límites del tablero, ambas están en diferentes posiciones y al menos una está activa.

# Ciclo 5. Nueva cosa: Proponiendo y diseñando

El objetivo de este punto es permitir recibir en una nueva cosa (no célula) en el replicate.

# (NO OLVIDE BDD – MDD)

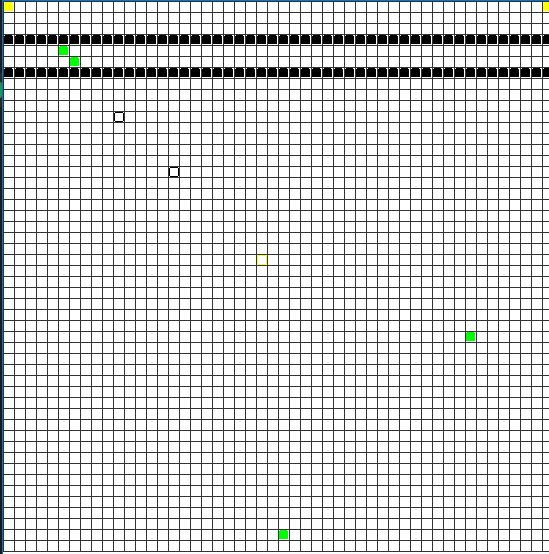
1. Propongan, describan e Implementen un nuevo tipo de Thing

(Mínimo dos pruebas de unidad)

La nueva cosa será un muro pegadizo (StickyWall). Si una célula es adyacente a un bloque de sticky Wall, la célula quedará pegada allí.

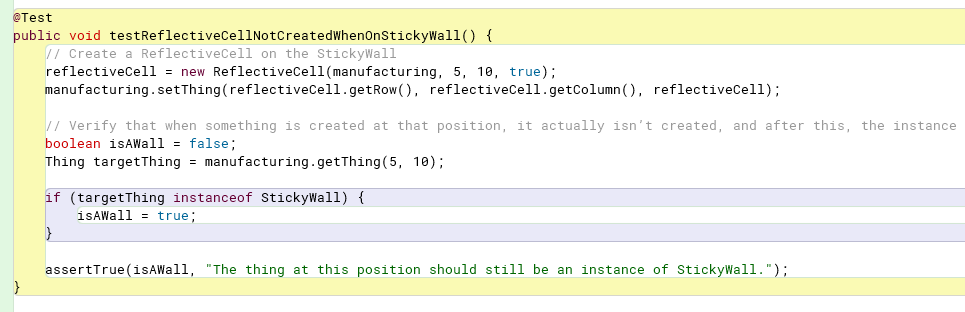
Si la célula es creada sobre un bloque de sticky wall, no se va a crear.

Para crear un sticky wall, bastará con proporcionar una fila, y el muro se creará a lo largo de esa fila. Se pueden crear tantos muros como se quiera.

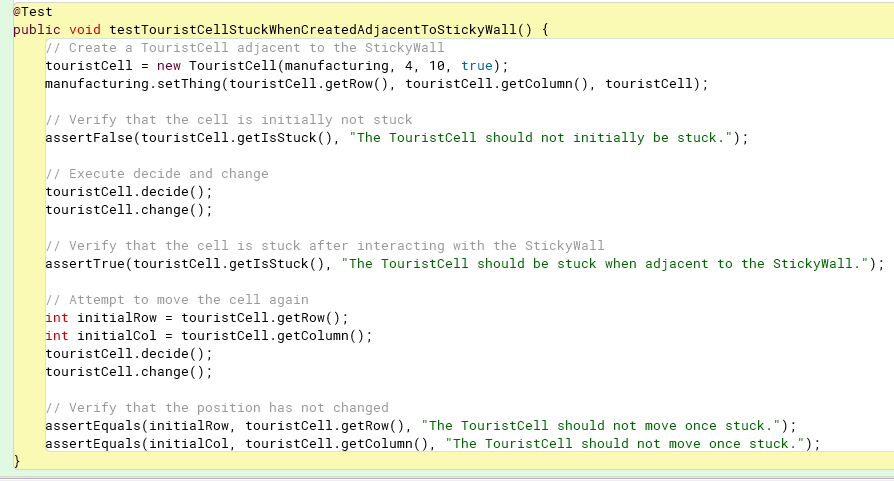


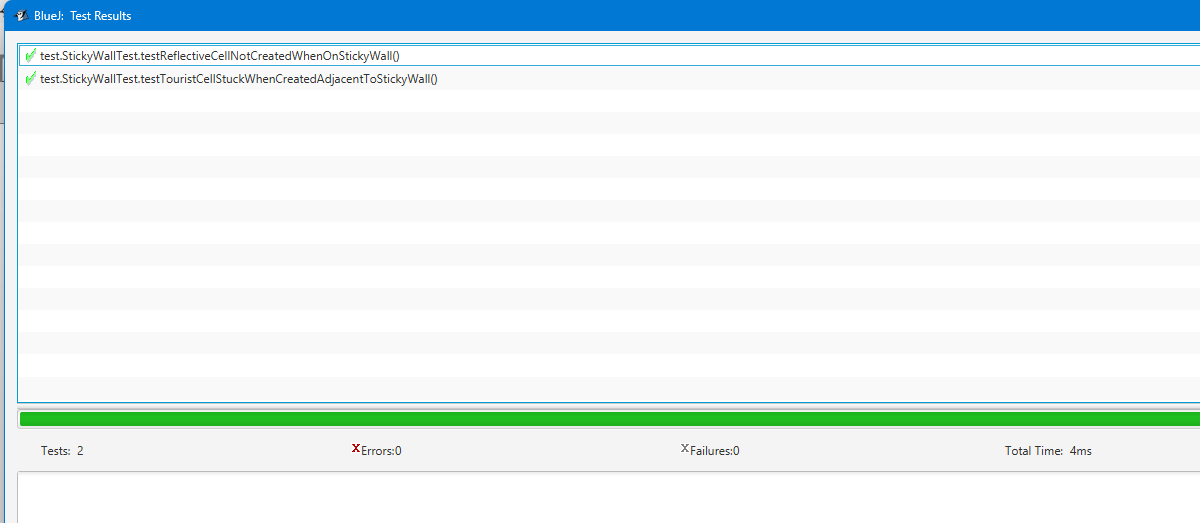
**Pruebas de unidad:**

1. **Una reflectiveCell es creada sobre el muro y en realidad no se crea**

****

1. **Una cell (touristcell) es creada abyacente a un stick wall y se comprueba que en efecto permanece pegada.**

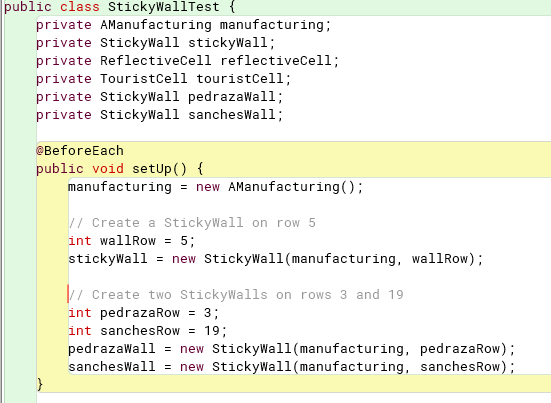
****

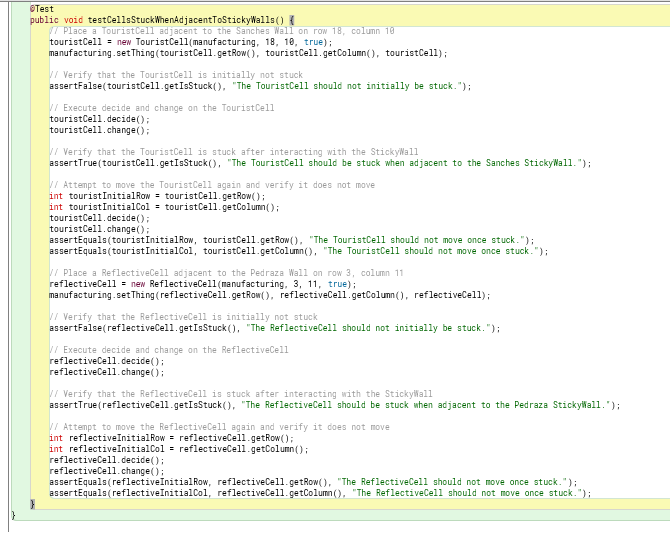


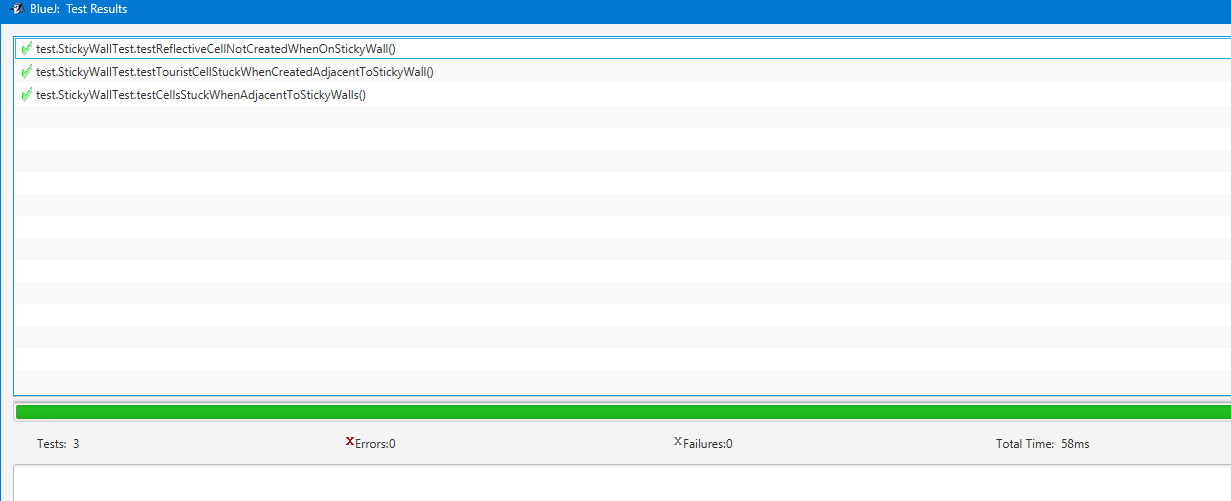
1. Considerando un par de ellos con los nombres de ustedes .
2. Piensen una prueba significativa y expliquen la intención.

Se crearan dos StickyWalls, pedraza y sanchez.  
Luego se crearan dos cells. Una touristCell adyacente a sánchez y una reflective cell adyacente a pedraza. En ambos casos, se comprobara que ambas permanecen pegadas.

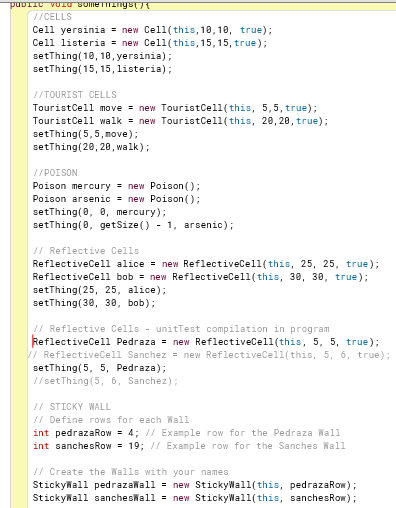
1. Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados.



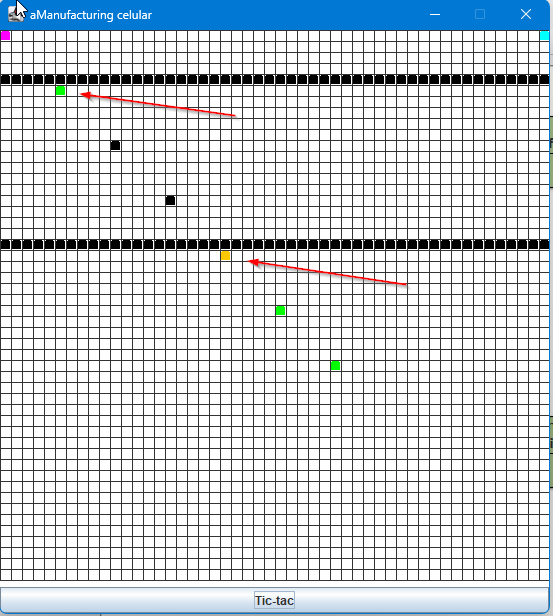




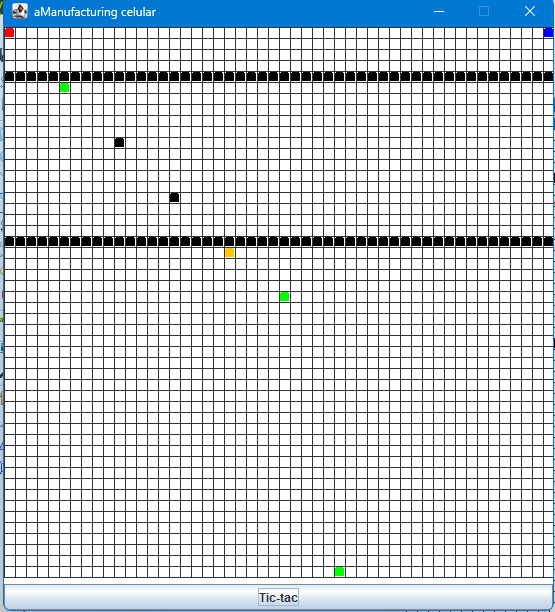
1. Ejecuten el programa con esa prueba para capturar las pantallas correspondientes.



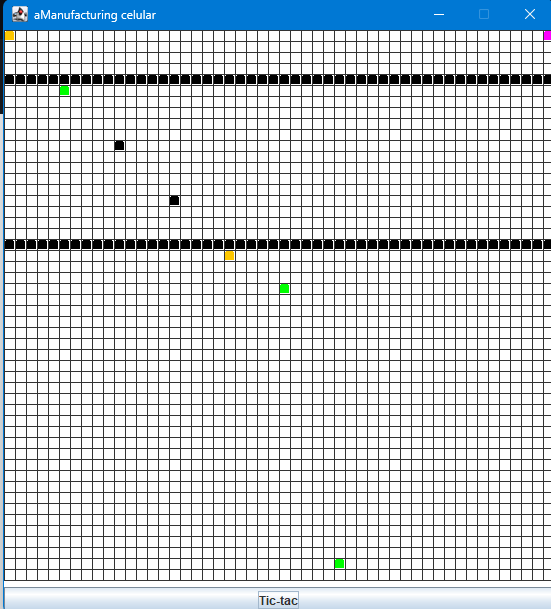
**INITIAL CONFIGURATION**

****

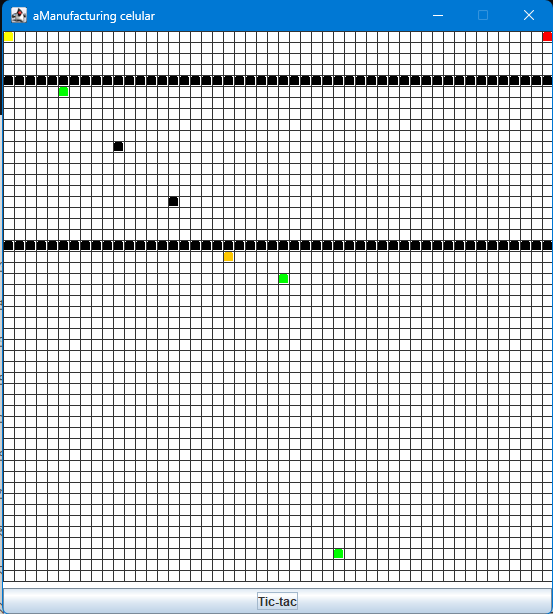
**FIRST MOMENT**

****

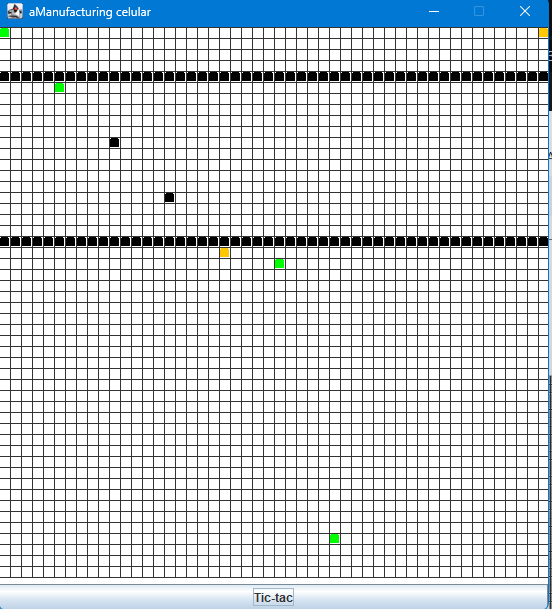
**SECOND MOMENT**

****

**THIRD MOMENT**

****

**FOURTH MOMENT**

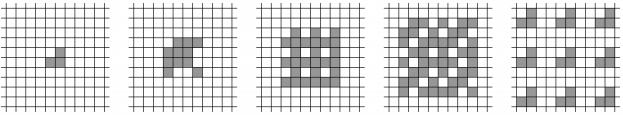
****

Como vemos, las dos permanecieron pegadas a sus stickywalls correspondientes. Resultando así en una prueba exitosa.

# Ciclo 6. BONO. Célula Original

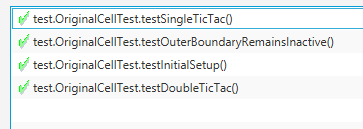
Una célula original es gris y si tiene un número impar de las nueve celdas vecinas activas, el estado en el paso siguiente es activa, de lo contrario estará inactiva.

Este sería el movimiento de las células originales en una zona de 10x10 que inicia llena de células inactivas salvo las tres del patrón central.



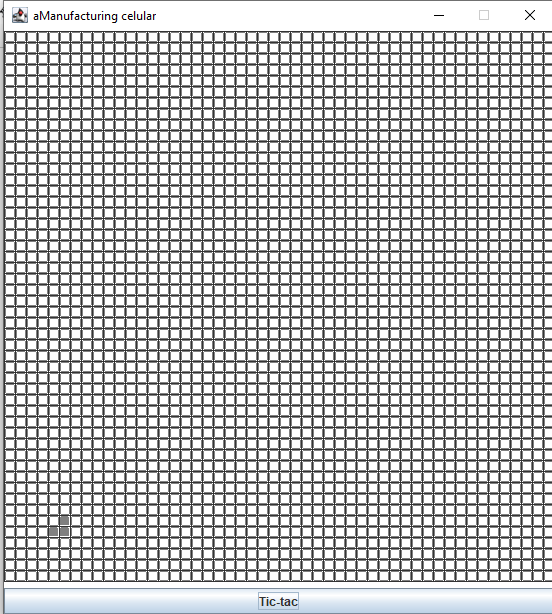
1. Diseñen ,construyan y prueben esta nueva clase.



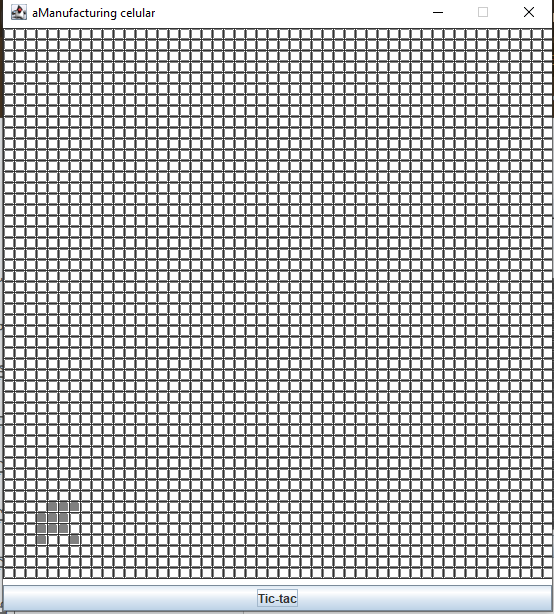


1. Inicien la esquina inferior izquierda con la configuración inicial y capturen las pantallas asociadas del ejemplo anterior.

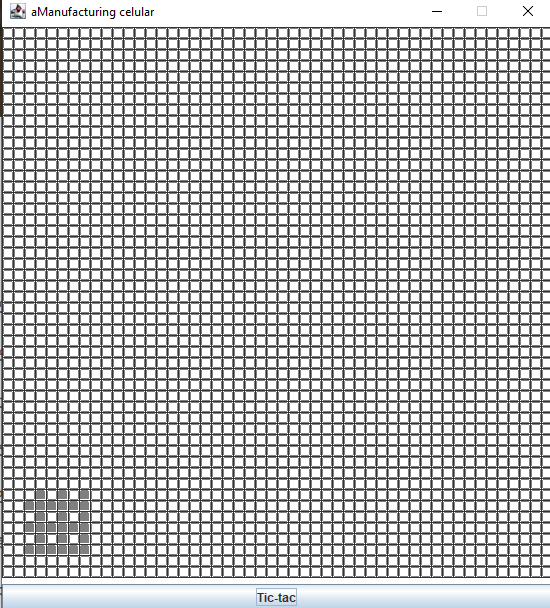
**INITIAL CONFIGURATION**

****

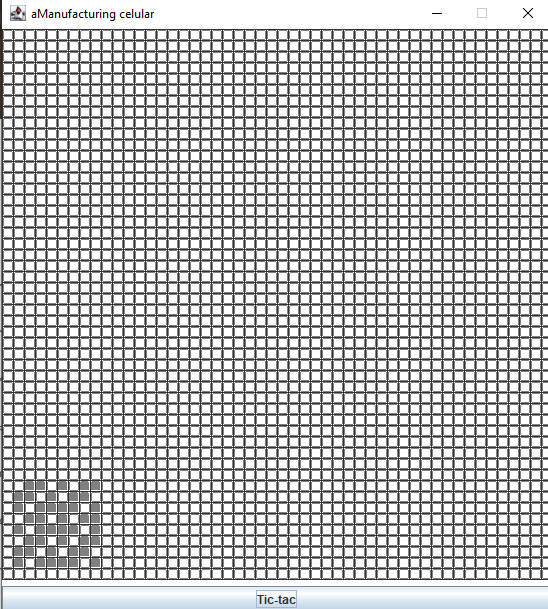
**FIRST MOMENT**

****

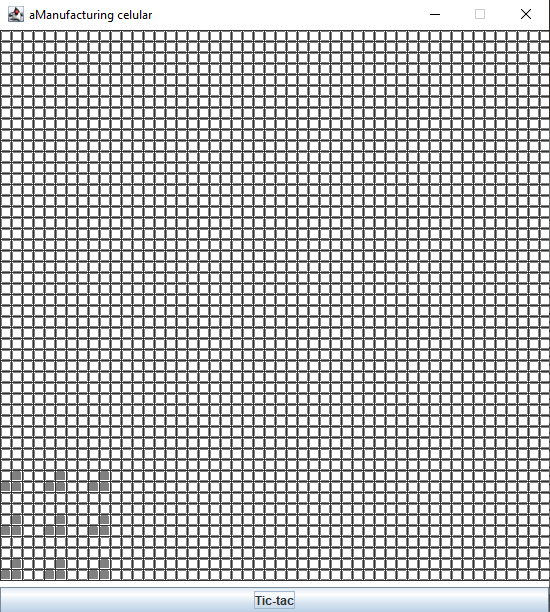
**SECOND MOMENT**

****

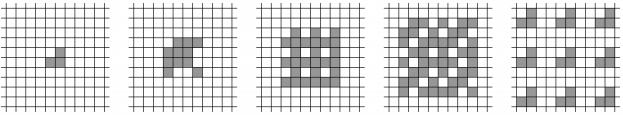
**THIRD MOMENT**

****

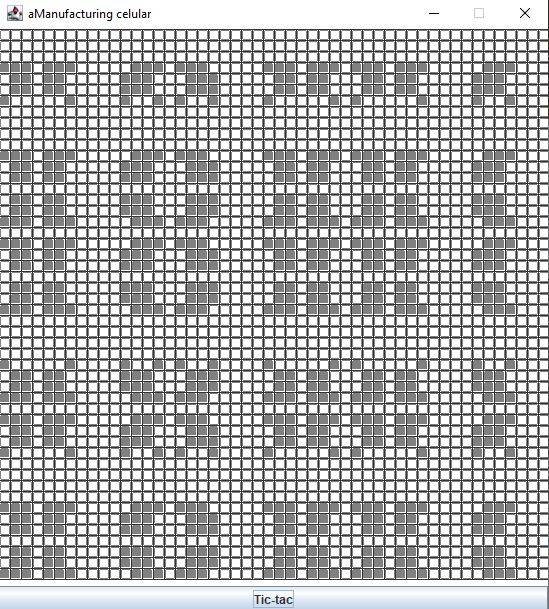
**FOURTH MOMENT**

****

Con esto, vemos que se cumplió con el requisito del bono que es llegar a obtener la imagen cuando se hace ticTac, partiendo desde la esquina inferior izquierda:

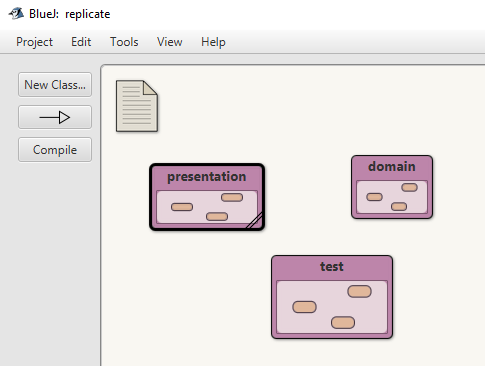


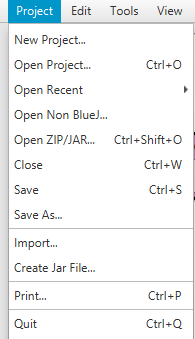
Después de muchos ticTac genera esto:

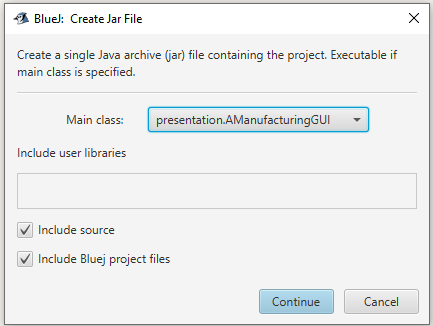


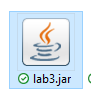
**Empaquetando la versión final para el usuario.** [En lab03.doc, replicate.asta , \*.java, replicate.jar]

1. Revise las opciones de BlueJ para empaquetar su programa entregable en un archivo .jar. Genere el archivo correspondiente.



 CLICK ON Create Jar File…





1. Consulte el comando java para ejecutar un archivo jar. ejecutennlo ¿qué pasa?

Para ejecutar un archivo JAR en Java, comando en la terminal:

java -jar nombre\_del\_archivo.jar

Java cargará el archivo JAR y comenzará a ejecutar la clase principal especificada en el manifiesto del JAR. Se abrirá una interfaz gráfica mostrando el “main” de AManufacturingGUI que es la solución al problema.



Se ve que se ejecutó el main de la clase principal, mostrando en este caso el bono(ciclo 6).

1. ¿Qué ventajas tiene esta forma de entregar los proyectos? Explique claramente.

* **Portabilidad**: El archivo JAR puede ejecutarse en cualquier máquina que tenga una JVM (Java Virtual Machine), sin necesidad de reconfiguración o compilación adicional.
* **Facilidad de distribución:** permite distribuir la aplicación como un único archivo.
* **Simplicidad en la ejecución:** un solo comando para convertir múltiples archivos en un solo para entregarlo como proyecto.
* **Estandarización:** archivos jar compatibles con muchas plataformas de distribución y despliegue.
* **Empaquetado de dependencias:** reducir conflictos o problemas con versiones de bibliotecas en la máquina del usuario.
* **Seguridad:** archivos jar pueden ser firmados digitalmente para verificar su autenticidad y garantizar que no hayan sido modificados.

# DE BLUEJ A CONSOLA

En esta sección del laboratorio vamos a aprender a usar java desde consola. Para esto se va a trabajar con el proyecto del punto anterior.

**Comandos básicos del sistema operativo** [En lab03.doc]

Antes de iniciar debemos repasar los comandos básicos del manejo de la consola.

1. Investiguen los comandos para moverse en la estructura de directorios: crear, borrar, listar su contenido y copiar o eliminar un archivo.

**Comandos bash - git bash**

### 1. Moverse en la estructura de directorios

* **cd (Change Directory):** Cambia el directorio actual.  
    
  cd [ruta\_del\_directorio]  
    
  cd /home/usuario/documentos
* **cd ..**: Sube un nivel en la estructura de directorios (al directorio padre).
* **cd ~**: Te lleva al directorio raíz de tu usuario (home).
* **pwd (Print Working Directory):** Muestra el directorio actual.

### 2. Crear un directorio

* **mkdir (Make Directory):** Crea un nuevo directorio.  
  mkdir [nombre\_del\_directorio]

### 3. Borrar un directorio

* **rmdir (Remove Directory):** Borra un directorio vacío.  
  rmdir [nombre\_del\_directorio]
* **rm -r (Remove Recursive):** Borra un directorio y su contenido (incluso si no está vacío).  
  rm -r [nombre\_del\_directorio]

### 4. Listar el contenido de un directorio

* **ls (List):** Lista los archivos y directorios dentro de un directorio.  
  + Opciones útiles:

**ls -l**: Muestra los archivos con detalles (permisos, propietario, tamaño, etc.).

**ls -a**: Muestra archivos ocultos.

### 5. Copiar un archivo

* **cp (Copy):** Copia archivos o directorios.  
    
  cp [ruta\_origen] [ruta\_destino]  
  cp archivo.txt /home/usuario/carpeta/

Para copiar un directorio completo: cp -r [directorio\_origen] [directorio\_destino]

### 6. Eliminar un archivo

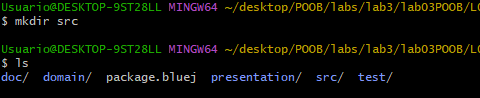
* **rm (Remove):** Elimina un archivo.  
  rm [nombre\_del\_archivo]  
  rm archivo.txt

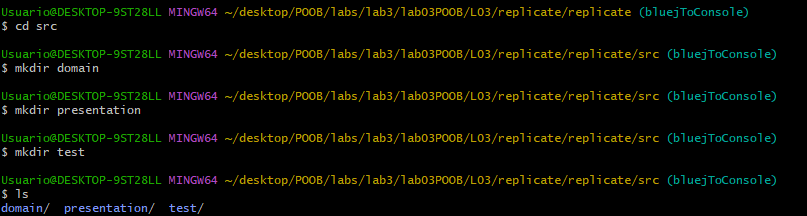
1. Organicen un nuevo directorio con la estructura propuesta para probar desde allí su habilidad con los comandos de consola. Consulten y capturen el contenido de su directorio

replicate

src

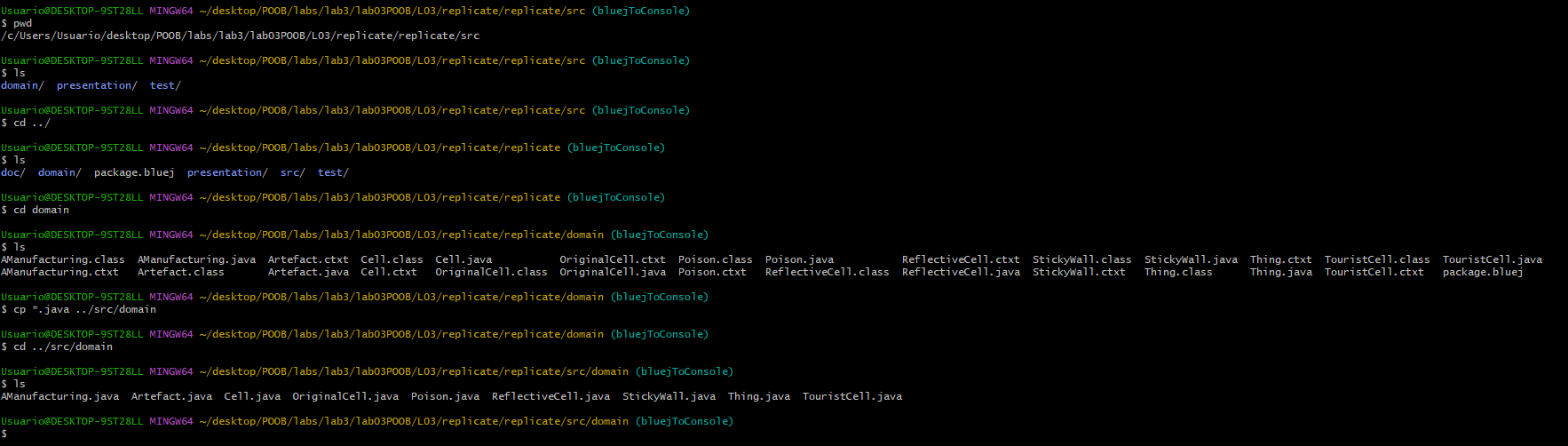
domain presentation test





1. En el directorio copien únicamente los archivos \*.java del paquete de aplicación .

Consulte y capture el contenido de src/domain





**Estructura de proyectos java** [En lab03.doc]

En java los proyectos se estructuran considerando tres directorios básicos.

replicate

src bin docs

1. Investiguen los archivos que deben quedar en cada una de esas carpetas y la organización interna de cada una de ellas.

En src debe quedar los archivos .java

En bin, deben quedar los .class, es decir, los archivos compilados.

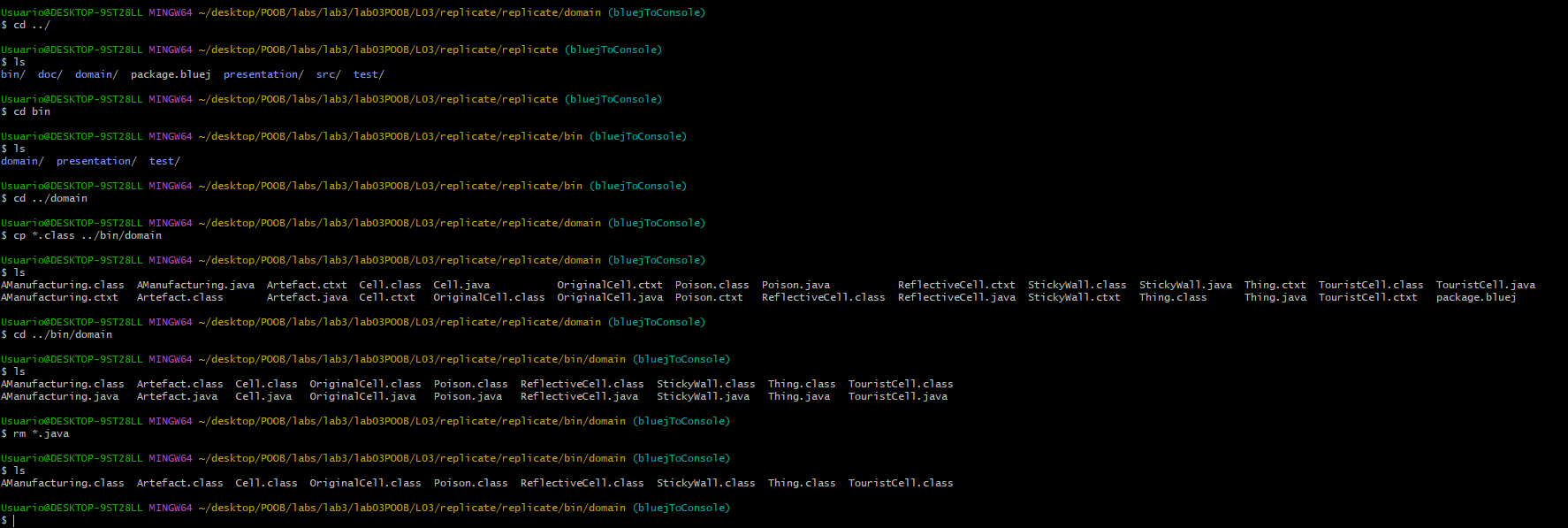
Y en docs se almacena la documentación generada.

La organización se orientada a paquetes, es el directorio/paquete/ y los archivos

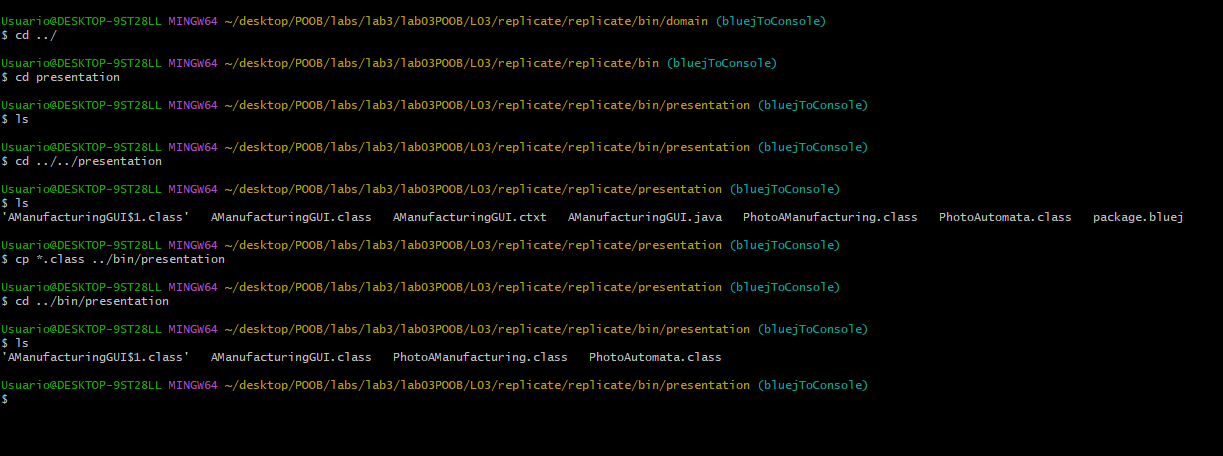
1. ¿Qué archivos debería copiar del proyecto original al directorio bin? ¿Por qué? Cópielos y consulte y capture el contenido del directorio que modificó.

Del proyecto original se deben copiar todos los .class.

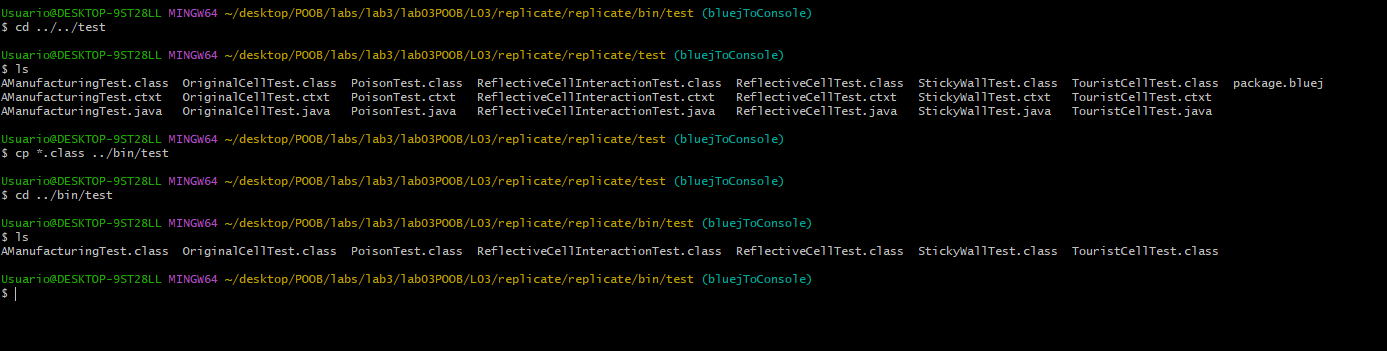
Añadiendo bin/domain



Añadiendo bin/presentation



Añadiendo bin/test



Ahora ubicaremos en sus respectivas carpetas los archivos .java (src)

**Comandos de java** [En lab03.doc]

1. Consulte para qué sirven cada uno de los siguientes comandos:

**javac:** Se utiliza para compilar archivos fuente de Java (.java) en archivos de bytecode (.class) que pueden ser ejecutados por la Java Virtual Machine (JVM).

**Sintaxis:**

javac NombreDelArchivo.java

**java:** Intérprete de la JVM, se usa para ejecutar archivos .class que contienen bytecode Java, es decir, el código que ha sido compilado, el comando carga el archivo .class y lo ejecuta dentro de la JVM.

**Sintaxis:**

java ClassName

**javadoc:** Se utiliza para generar documentación en formato HTML a partir de comentarios de documentación en el código fuente de Java. Esta documentación es especialmente útil para describir clases, métodos y atributos.

**Sintaxis:**

javadoc FileName.java

**jar:** Los archivos JAR son contenedores que pueden incluir múltiples archivos .class, imágenes, bibliotecas y otros recursos. Se utilizan para empaquetar aplicaciones completas en un único archivo distribuible.

**Sintaxis:**

* Crear un archivo JAR:

jar cvf NombreDelArchivo.jar archivo1 archivo2 …

* Extraer un archivo JAR:

jar xvf NombreDelArchivo.jar

* Ver el contenido de un archivo JAR:

jar tvf NombreDelArchivo.jar

1. Cree una sesión de consola y consulte en línea las opciones de los comandos java y

javac. Capture las pantallas.

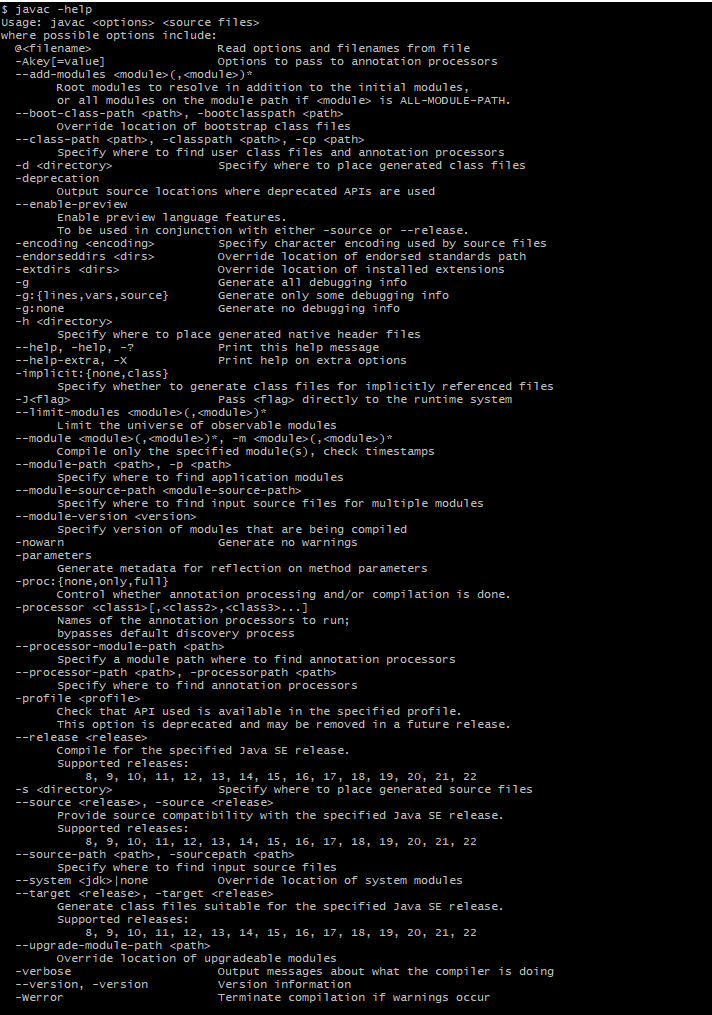
Con los comandos java -help y javac -help se muestran las opciones que tiene cada una de los comandos

**Con java -help**



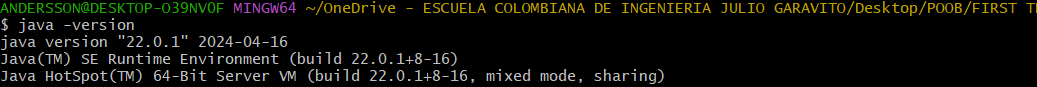


**Con javac -help**

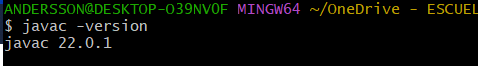


1. Busque la opción que sirve para conocer la versión a qué corresponden estos dos comandos. Documente el resultado.

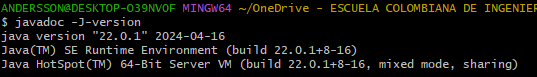
java -version



javac -version



javadoc -version



**Documentación del Resultado**

Ambos comandos (java -version y javac -version) proporcionan información importante sobre las versiones de Java instaladas. Esto es especialmente útil para desarrolladores, ya que algunas características y compatibilidades del lenguaje pueden variar entre versiones.

Conocer la versión exacta ayuda a:

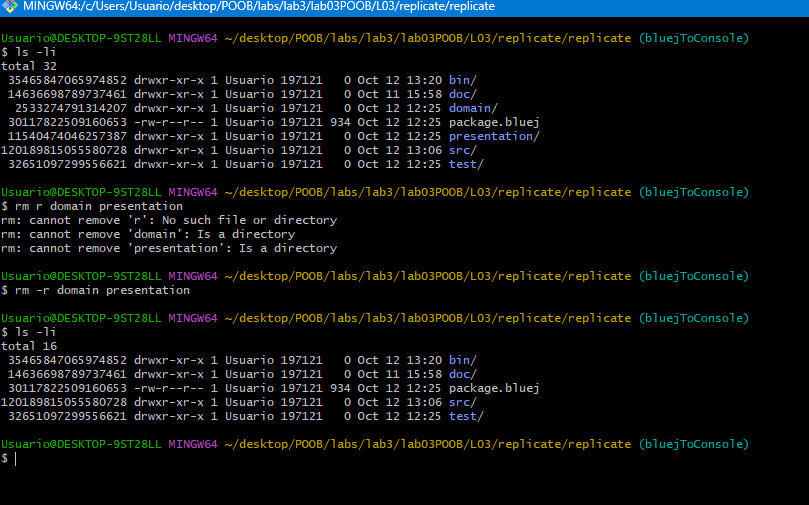
* Asegurar la compatibilidad entre el código y el entorno de ejecución.
* Verificar la configuración del entorno de desarrollo.
* Diagnosticar problemas de compatibilidad.

Documentando estos comandos, se puede confirmar que tanto el JRE como el JDK están actualizados y correctamente instalados para proyectos en Java.

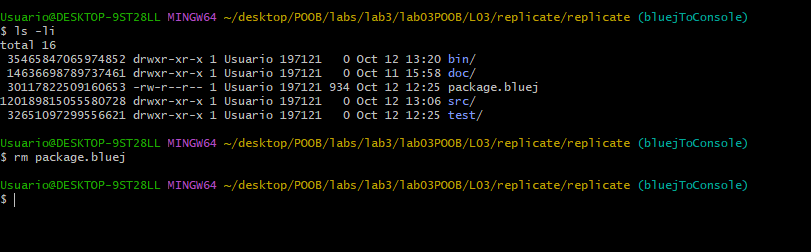
**Compilando** [En lab03.doc]

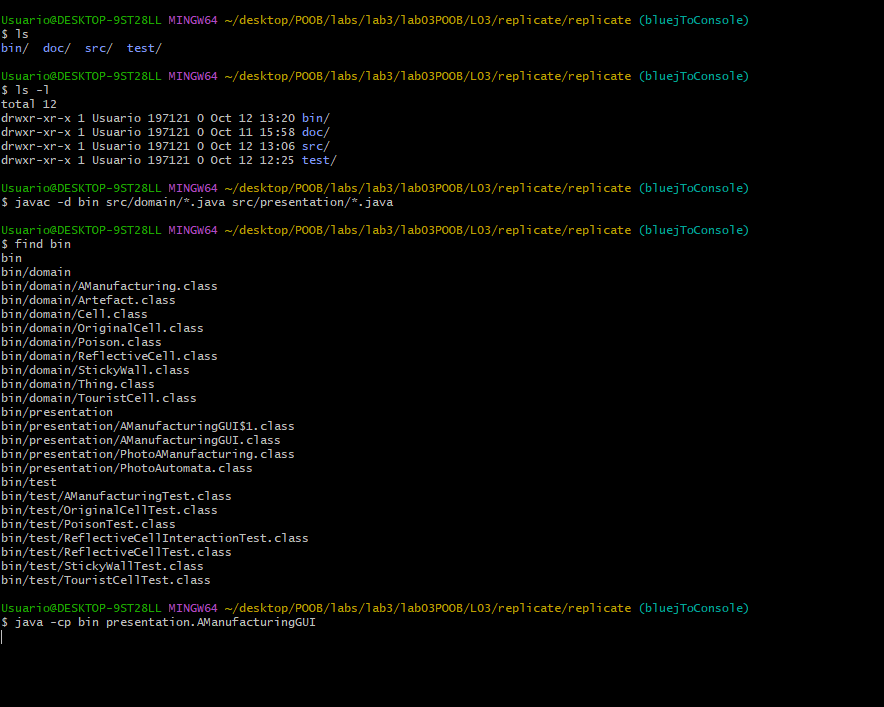
1. Utilizando el comando javac, **desde el directorio raíz (desde replicate con una sóla instrucción)**, compile el proyecto.

Dado que todo quedó organizado en directorios, vamos a eliminar los directorios de dominio y presentation dado que ya están en src



Y ya no necesitamos package.bluej

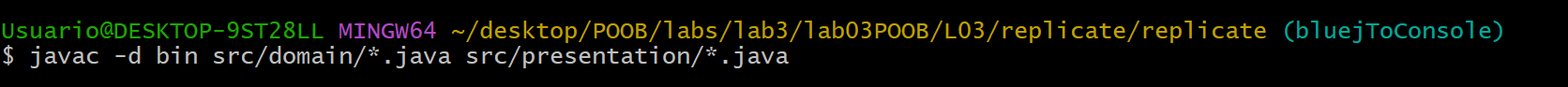




¿Qué instrucción completa tuvo que dar a la consola para compilar TODO el proyecto? Tenga presente que se pide un único comando y que los archivos compilados deben quedar en los directorios respectivos.

El comando completo fue

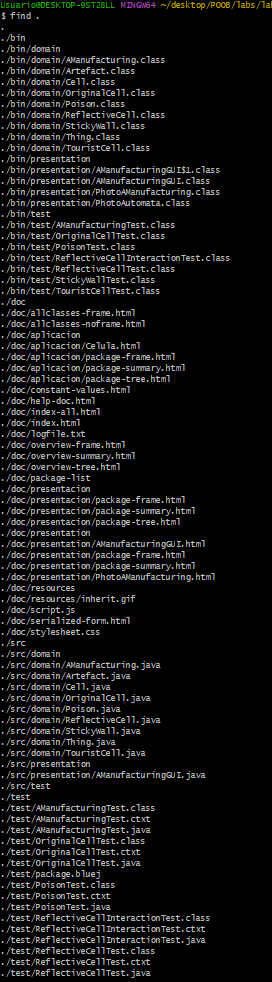
$ javac -d bin src/domain/\*.java src/presentation/\*.java



En donde se le dice que ejecute los java pero que guarde los compilados en bin

1. Revise de nuevo el contenido del directorio de trabajo y sus subdirectorios. ¿Cuáles nuevos archivos aparecen ahora y dónde se ubican?

Revisando todos los directorios y subdirectorios con find .

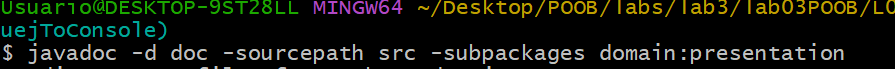


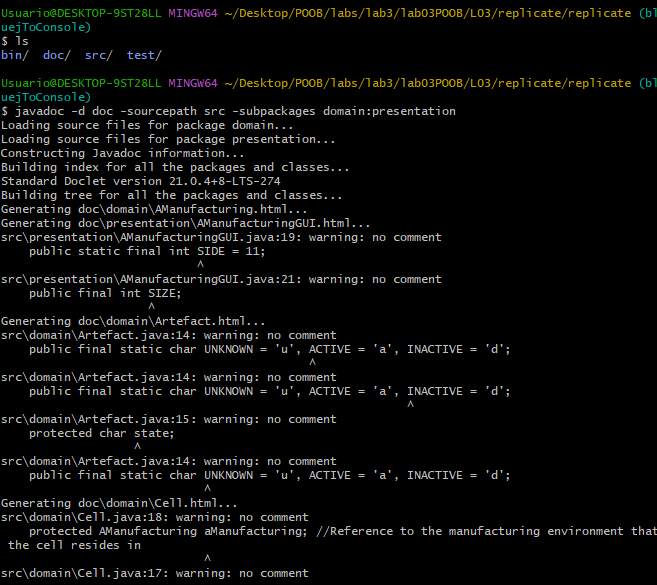
Encontramos que los nuevos archivos son los .class en en path bin especificado.

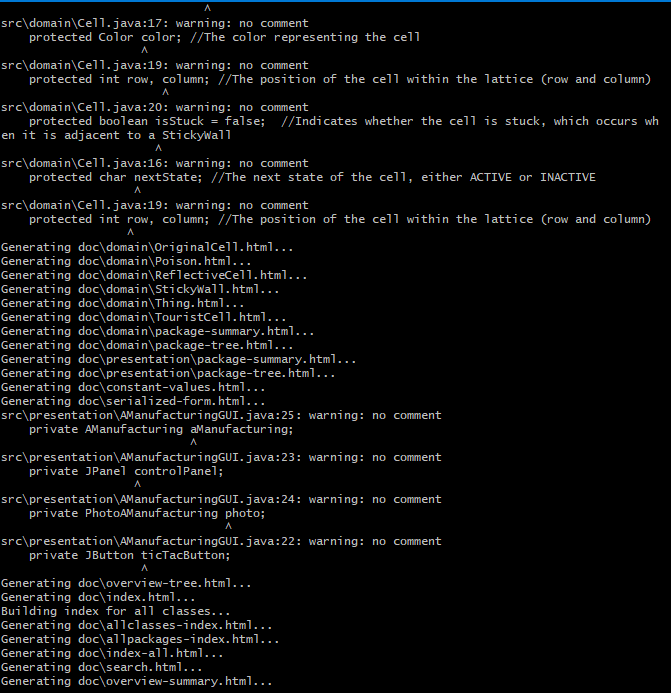
**Documentando** [En lab03.doc]

1. Utilizando el comando javadoc, desde el directorio raíz, genere la documentación (API) en formato html, en este directorio. ¿Cuál es el comando completo para generar esta documentación?

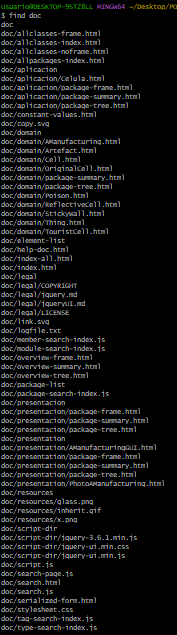
El comando que se utilizó es el siguiente.

  
  
Se utilizó dado que queremos que se documenten los paquetes y subpaquetes de src, pero no todos, solo los que especificamos, todos menos test.

Y se ejecutó de la siguiente manera  


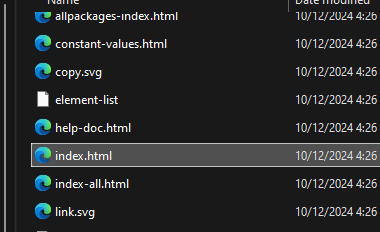


El directorio doc ha sido modificado, teniendo la siguiente estructura

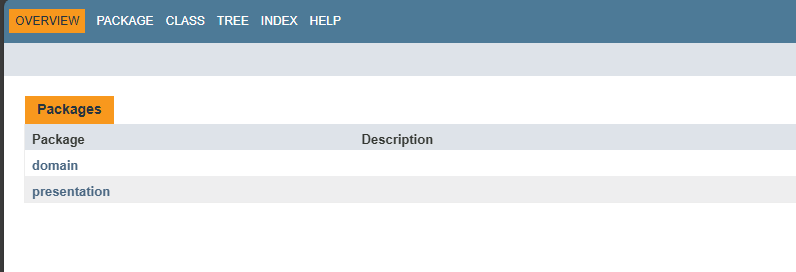


1. ¿Cuál archivo hay qué abrir para empezar a navegar por la documentación? Ábralo y capture la pantalla.

El archivo se debe abrir primero es index.html.



En este caso solo estan los paquetes de domain y presentation dado que no se nos pidió que añadiremos test



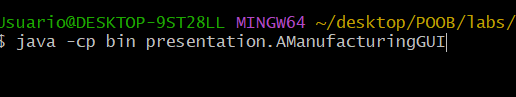
**Ejecutando** [En lab03.doc]

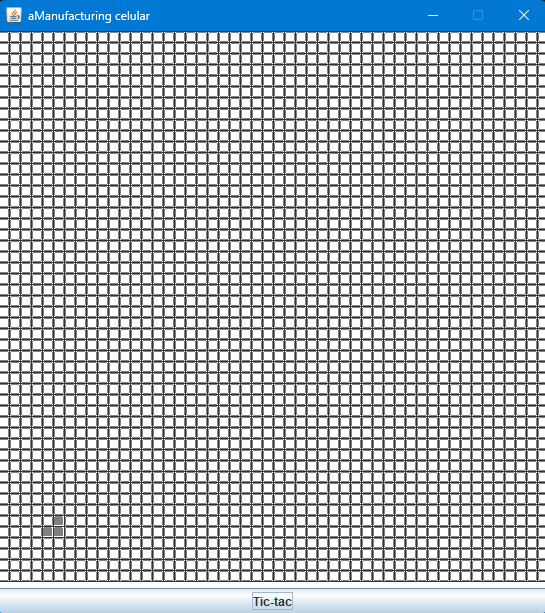
1. Empleando el comando java, desde el directorio raíz, ejecute el programa. ¿Cómo utilizó este comando?

Empleando el comando java, tenemos que posicionarnos en la raíz del proyecto.

Una vez allí, identificamos la clase principal (con el main) y le diremos al JVM que tiene que buscar los archivos .class que necesita en el directorio bin.

**java -cp bin presentation.AManufacturingGUI**  
  
 El comando general queda así

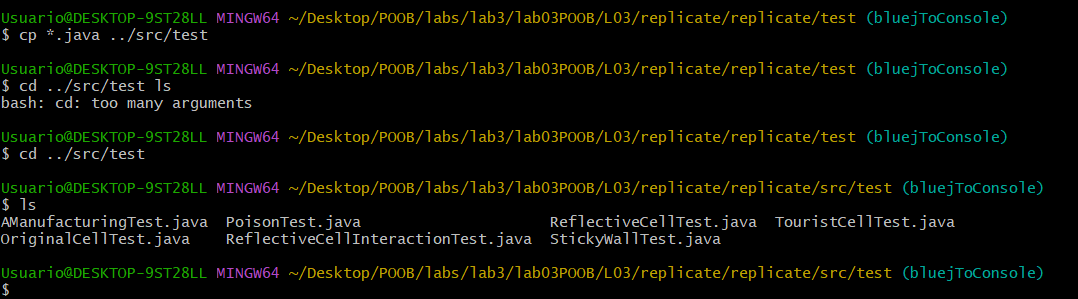




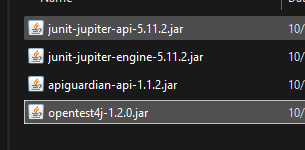
**Probando** [En lab03.doc]

1. Adicione ahora los archivos del directorio pruebas y trate de compilar nuevamente el programa.Tenga en cuenta que estas clases requieren la librería junit 4.8.

Adicionando los archivos al directorio de pruebas



Fue necesario ir a marven y descargar cuatro jar



Se utilizó JUnit5, y dado que había una clase de prueba unitaria que utiliza JUnit4, se actualizó a JUnit5.

**¿Cómo se incluye un paquete para compilar?**

El paquete se incluyó añadiendo al path src/test/\*.java.

Pero como son pruebas, y utilizan librerías de JUnit, se guardaron en la carpeta libs y en el path se llamaron con libs/\*.jar

**¿Qué instrucción completa tuvo que dar a la consola para compilar?**

javac -cp ".;libs/\*" -d bin src/domain/\*.java src/test/\*.java



1. [Ejecute desde consola las pruebas . ¿Cómo utilizó este comando?.](https://stackoverflow.com/questions/2235276/how-to-run-junit-test-cases-from-the-command-line)

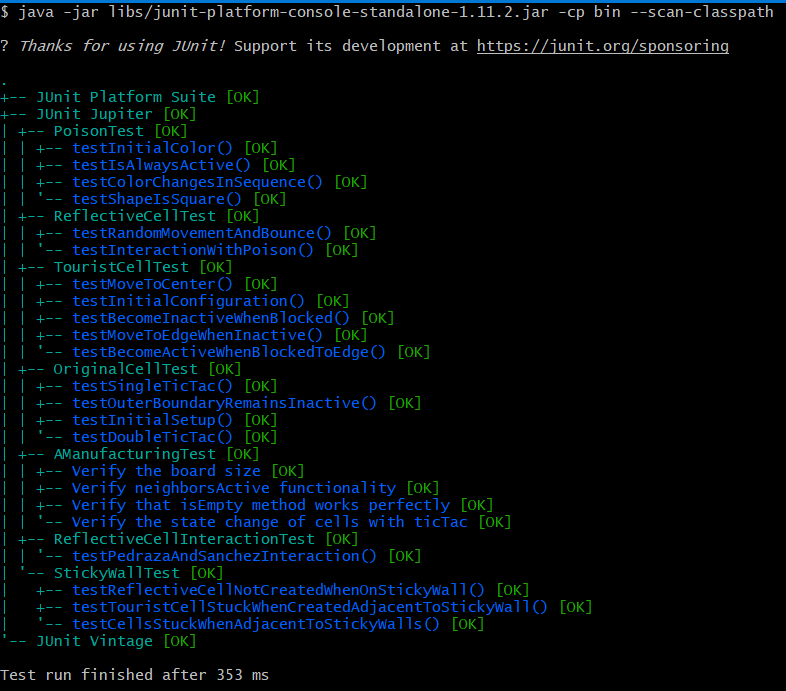
Para ejecutar las pruebas, descargamos el jar



Y ejecutamos el comando

$ java -jar libs/junit-platform-console-standalone-1.11.2.jar -cp bin --scan-classpath  


1. Pegue en su documento el resultado de las pruebas

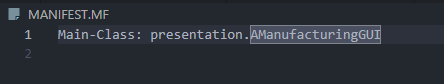


**Empaquetando** [En lab03.doc]

1. Consulte cómo utilizar desde consola el comando jar para empaquetar su programa entregable en un archivo .jar, que contenga los archivos bytecode necesarios (no las fuentes ni las clases de prueba), y que se pueda ejecutar al instalarlo en cualquier directorio, con solo tener la máquina virtual de java y su entorno de ejecución (JRE).

Primero definimos la main-class dentro del archivo manifiesto





**¿Cómo empaquetó jar ?**

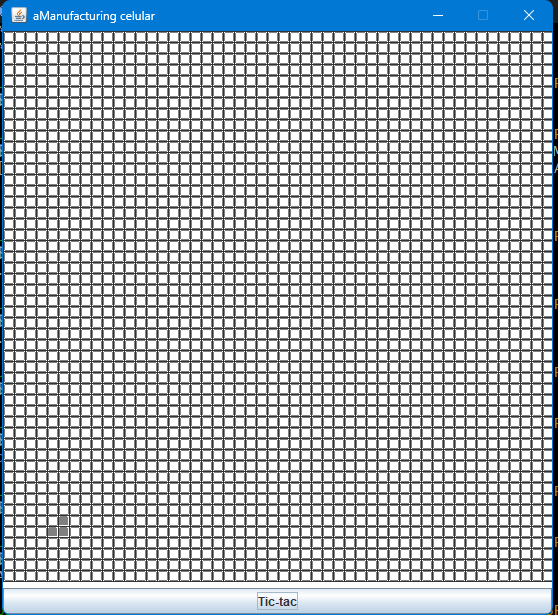
Con este comando  
  
$ jar cfm AManufacturingApp.jar MANIFEST.MF -C bin .





1. ¿Cómo se ejecuta el proyecto empaquetado?





# RETROSPECTIVA

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

Fue de 24 horas trabajando siempre en equipo durante varios días de poco a poco.

1. ¿Cuál es el estado actual del laboratorio? ¿Por qué? (Para cada ciclo incluya su estado)

El estado del lab es terminado completamente porque el trabajo en equipo, y la buena investigación hicieron que cumplieramos con los objetivos.

**CICLO 1** Solo entender el código que ya estaba establecido e implementar ticTac.

**CICLO 2** Células turistas fue interesante el cambio de color cuando cambia de un estado a otro.

**CICLO 3** Crear clase Poison fue fácil, lo único que tocó investigar es lo del tema aleatorio.

**CICLO 4**  Crear un nuevo tipo de célula nos puso a definir reglas que usan clase definidas anteriormente.

**CICLO 5** Poner en la mesa la creatividad para inventar un nuevo requisito (Thing) fue enriquecedor

**CICLO 6** Fue gratificante cuando se ejecutó el main que diera lo mismo que tenía que dar

**Empaquetando la versión final para el usuario** Fue fácil, solo era encontrar la opción en BlueJ para generar el archivo jar, y luego ejecutar el comando.

**BlueJ a Terminal:** consideramos que algunas partes tocó investigar más que otras, pero se lograron cumplir todos los requisitos.

1. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importantes?

Las prácticas de Extreme Programming (XP), como escribir pruebas unitarias primero y seguir estándares de código, nos feron útiles para asegurar calidad desde el inicio, facilitar el mantenimiento, reducir errores y costos de depuración, y promover una colaboración efectiva.Nos permitieron desarrollar software robusto, adaptable y comprensible para todo el equipo.

1. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?

El mayor logro fue terminar el laboratorio porque estaba largo. El mayor problema que hubo fue en detectar que toca instalar java correctamente, en el sentido que javadoc no nos funcionaba, y era porque el path de Java no estaba en el directorio correspondiente en environment variables; también ejecutar desde el terminal las pruebas, se tuvo que recurrir a instalar varios archivos de JUnit para que funcionara de manera adecuada.

La solución de estos problemas fue la investigación constante hasta lograr ejecutar las pruebas.

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

Como equipo, hicimos un buen trabajo al colaborar en Pair Programming, compartiendo ideas y detectando errores en tiempo real, lo que mejoró la calidad del código y aumentó nuestra eficiencia. Para mejorar, nos comprometemos a fortalecer la comunicación y a revisar juntos las mejores prácticas antes de comenzar cada tarea, asegurando un enfoque más consistente y alineado con nuestros objetivos.