**Laboratorio 03. POOB-02**

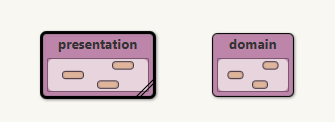
Jefer Alexis González Romero  
Angel Nicolas Cuervo Naranjo

**Conociendo**

**1.** En el directorio descarguen los archivos contenidos en replicate.zip. Revisen el código de la aplicación

**a)** ¿Cuántos paquetes tiene?

Tiene dos paquetes: *Presentation* y *Domain.*



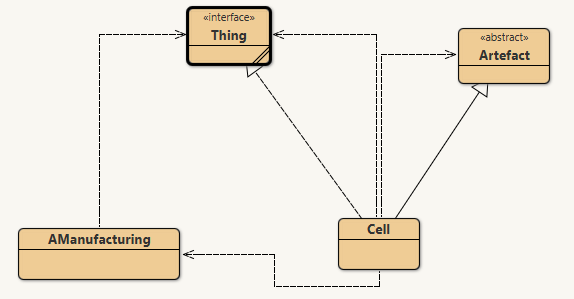
**d)** ¿Cuál es el propósito del paquete presentación?

Su propósito es generar un entorno que permita a los usuarios interactuar con el programa.

**e)** ¿Cuál es el propósito del paquete dominio?

Dominio tiene como propósito crear la red bidimensional de celdas y definir las características de los elementos presentes, entre estos el de las células.

**2.** Revisen el paquete de dominio,



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

Tiene componentes de tipo *interface,* clases concretas y clases abstractas.

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

El tipo *interface* implica que no puede ser instanciada y permite hacer herencia múltiple para definir el comportamiento de una clase.

Las clases concretas implica que si se pueden instanciar objetos de esta.

Las clases abstractas implican que no se puedan instanciar objetos de estas clases y deben contener al menos un método abstracto.

**3.** Revisen el paquete de dominio,

**a)** ¿Cuántos componentes tiene?

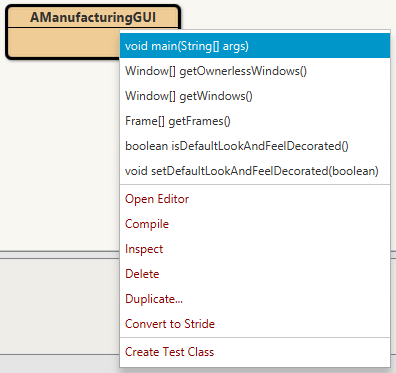
Tiene 4 componentes.

**b)** ¿Cuántos métodos ofrece?

Ofrece 24 métodos públicos, uno privado y uno protegido

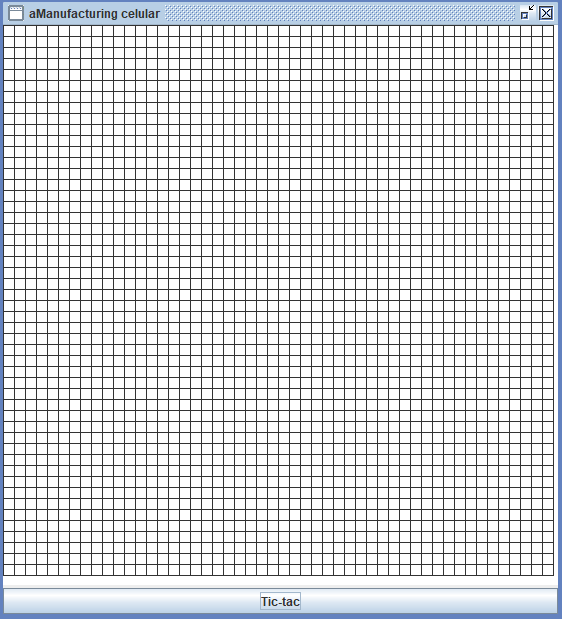
**4**. Para ejecutar un programa en java, ¿Qué método se debe ejecutar? ¿En qué clase de replicate se encuentra?

Para ejecutar programas en java se debe ejecutar el método *main*, este es un estándar utilizado por la *JVM* para iniciar sus ejecuciones. En *replicate* se encuentra en la clase *AManufacturingGUI.*



**5.** Ejecuten el programa. ¿Qué funcionalidades ofrece? ¿Qué hace actualmente? ¿Por qué?

Al ejecutar el método *main* de *AManufacturingGUI,* ofrece el método Tic-Tac que por ahora no hace nada, debido a que aún no se ha hecho su implementación.



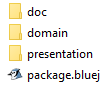
**Arquitectura general.**

**1.** Consulten el significado de las palabras package e import de java. ¿Qué es un paquete? ¿Para qué sirve? Explique su uso en este programa.

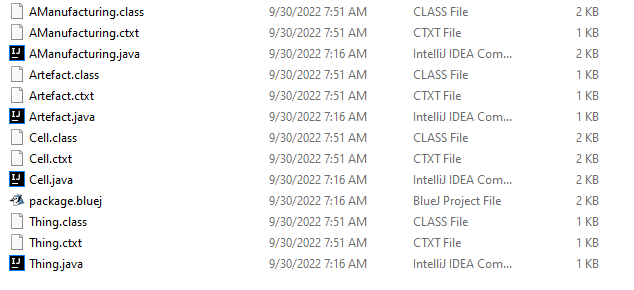
*Package* en java es donde se agrupan las clases relacionadas para facilitar la modularidad del código. *Import* se usa para poder acceder a una clase o interfaz de otro paquete.

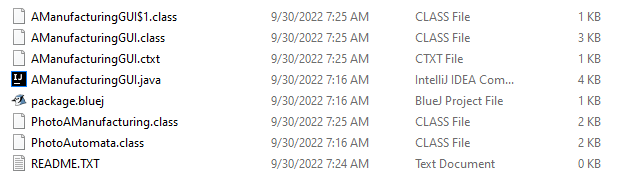
El uso de los paquetes en *replicate* nos sirven para dividir las clases relacionadas con la interacción de los usuarios y las clases que implementan el programa. Además, de que nos permite hacer uso de *domain* en *presentation.*

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



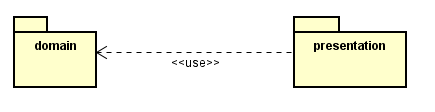
*Replicate* contiene 3 directorios: *doc* que contiene la documentación del programa*, domain* donde están las clases relacionadas con la implementación del programa*, presentation* que tiene las clases relacionadas con la interacción del usuario con el programa y un archivo llamado *package* con extensión *.bluej.*





Los directorios conservan el nombre del paquete y contiene todo lo relacionado con las clases contenidas en el paquete.

**3.** Adicionen 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.



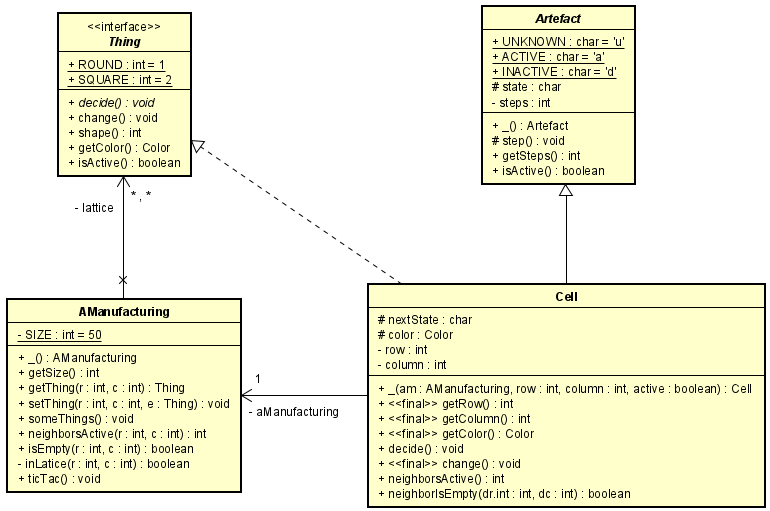
En replicate.astah

**Arquitectura detallada.**

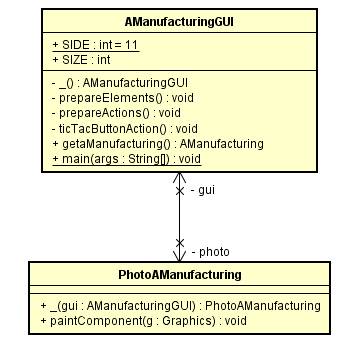
**1.** Para preparar el proyecto para BDD. Completen el diseño detallado del paquete de dominio.

**a)** ¿Qué componentes hacían falta?

Hacía falta la clase concreta *Cell.*



**2.** Revisen 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?



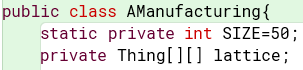
Solo hay un archivo .java, ya que PhotoAManufacturing al no ser publico solo puede ser usada dentro del paquete que la contiene, así que no es necesario crear un archivo .java para esta.

**3.** Adicionen en las fuentes la clase de pruebas unitarias necesaria para BDD. (No lo adicione al diagrama de clases) ¿a) En qué paquete debe estar? b) ¿Asociado a qué clase? c) ¿Por qué?

Debe estar en el paquete *dominio*, debido a que en este se encarga de la funcionalidad del programa y está asociado a la clase *AManugfacturing,* porque esta es la clase principal y es la que contiene los elementos del programa.

**Ciclo 1. Iniciando con las células normales**

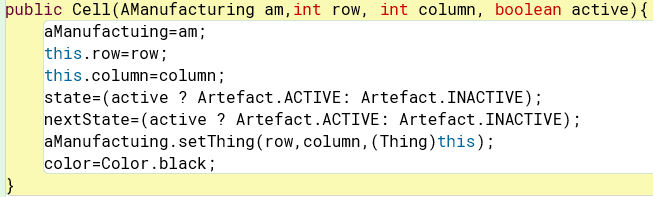
**1.** Estudie la clase AManufacturing ¿Qué tipo de colección usa para albergar las Thing? ¿Puede recibir células? ¿Por qué?



Usa un *array* de dos dimensiones llamado *lattice* para almacenar las *Thing*. Si puede recibir las células, ya que estas implementan a la interfaz Thing.

**2.** Estudie el código asociado a la clase Cell,

**a)** ¿en qué estado se crea?



El estado de creación depende de que reciba en el parámetro *active,* si recibe *true,* entonces el estado será ‘a’ que representa activo. Por otro lado, si active es *false,* el estado será ‘d’ que significa inactivo.

**b)** ¿qué forma usa para pintarse?

En interfaz *Thing:*

Texto

Descripción generada automáticamente con confianza baja

En clase *PhotoAManufacturing:*

Texto

Descripción generada automáticamente

Como implementa la interfaz *Thing* y no tiene sobre escrito el método shape()*,* siempre cuando se va a pintar *shape()* por defecto va a retornar *SQUARE* = 2 que representa la forma cuadrada.

**c)** ¿cuándo cambia de etapa?

En clase *Cell:*

Texto

Descripción generada automáticamente

En clase *Artefact:*

Imagen que contiene Diagrama

Descripción generada automáticamente

Cambia cuando se invoca el método *change(),* que llama al método *step(),* el cual suma uno en el atributo *steps.* Después le asigna a *state* el valor de *nextState*.



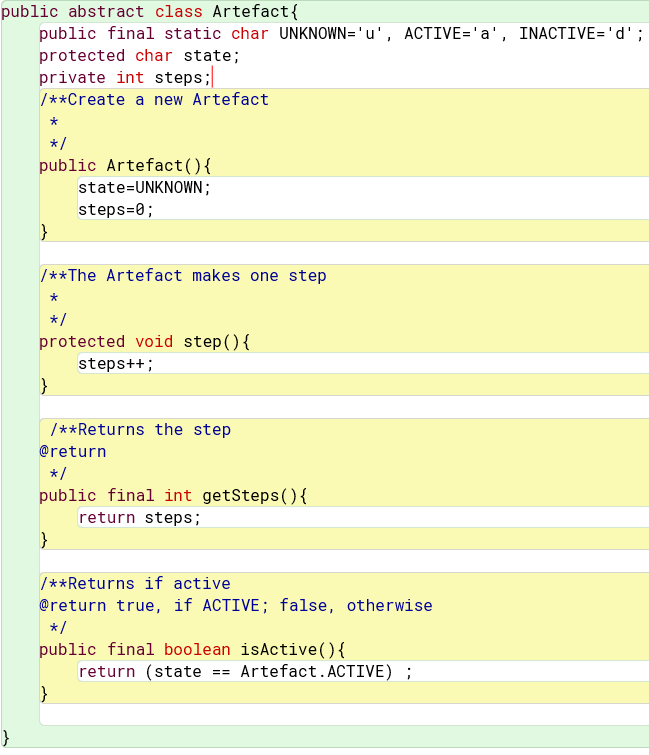


El valor de *nextState* al momento de creación de la célula es el mismo que tiene *state,* ya después se cambia el valor de este con *decide(),* el cual dependerá del número que tiene *steps,* si es múltiplo de 3 quedará en estado activo, de lo contrario, en inactivo.

**d)** ¿qué componentes la definen?

La clase *Cell* la definen dos componentes: la interfaz *Thing* y la clase abstracta *Artefact,* ya que esta la implementa y la extiende respectivamente, por lo tanto, definen algunos estados y comportamiento de *Cell.*

**3.** Cell por ser un Artefact,



**a)** ¿qué atributos tiene?

Por ser un *Artefact,* tiene tres atributos finales y estáticos de tipo *char: UNKNOWN, ACTIVE e INACTIVE*, un atributo protegido de tipo *char* que es *state y* un atributo privado de tipo *int* que se llama *steps.*

**b)** ¿qué puede hacer (métodos)?

*artefact():* Crea un nuevo artefacto.

*step():* Dar un paso, sumándole a *steps* uno.

*getSteps():* Consultar su atributo *steps.*

*isActive:* Decir si está activo.

**c)** ¿qué decide hacer distinto (métodos sobreescritos)?

No tiene métodos sobre escritos de *Artefact.*

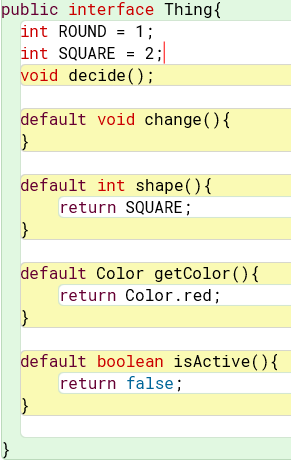
**d)** ¿qué no puede hacer distinto (métodos finales)?

No puede obtener los pasos de otra manera (*getSteps()),* y no puede decir si está activo de forma distinta (*isActive()).*

**e)** ¿qué debe aprender a hacer (métodos abstractos)?

*Artefact* no tiene métodos abstractos.

**4.** Por comportarse como un Thing,



**a)** ¿qué atributos tiene?

Por comportarse como un *Thing, t*iene dos atributos de tipo *int: ROUND y* SQUARE.

**b)** ¿qué puede hacer (métodos)?

*decide():* Decidir su siguiente estado.

*shape():* Consultar su forma.

*getColor():* Consultar su color.

*isActive:* Decir si está activo.

**c)** ¿qué decide hacer distinto (métodos sobreescritos)?

Decide hacer distinto el método *getColor()* para consultar su color y cambia la forma de decidir el siguiente estado (*decide()) .*

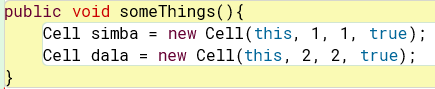
**d)** ¿qué no puede hacer distinto (métodos finales)?

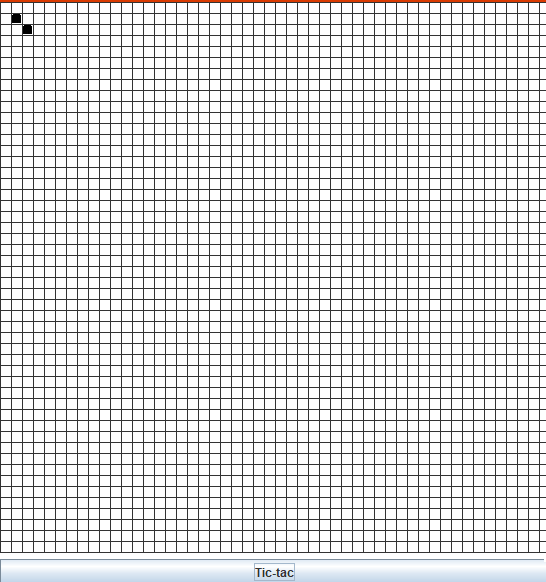
*Thing* no tiene métodos finales.

**e)** ¿qué debe aprender a hacer (métodos abstractos)?

El único método abstracto que debe aprender es *decide()* para decidir su próximo estado*.* Por los demás (*change(), shape(), getColor(), isActive())* puede elegir aprenderlos o no.

**5.** Ahora vamos a crear dos células activas en diferentes posiciones (1,1) (2,2) llámelos simba y dala usando el método someThings() . Ejecuten el programa,





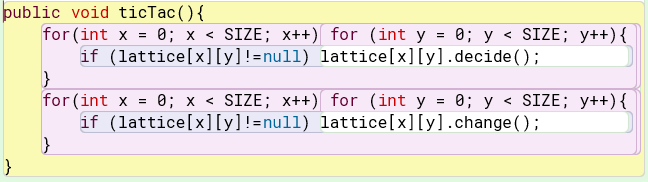
**a)** ¿Qué pasa con las células?

Son ubicadas con forma rectangular dentro de la red bidimensional de celdas en las posiciones especificadas, con estado activo, representado con el color negro.

**b)** ¿Por qué?

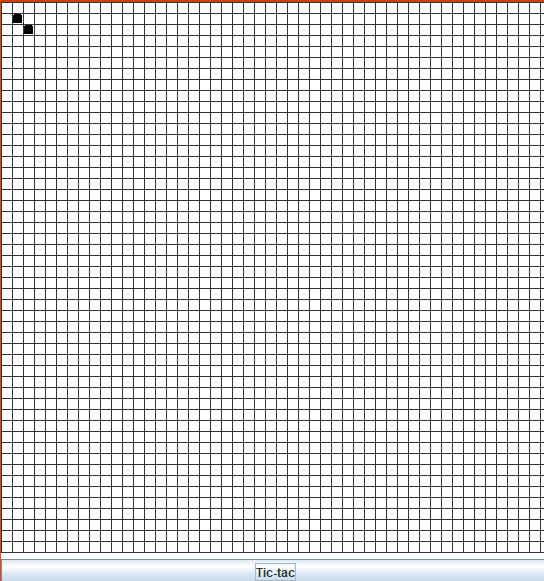
Porque si el estado de las células es activo las representa con forma cuadrada (por la interfaz *thing)*, con contorno y relleno de color negro que es el que se les asigna a todas las células al crearse. Están en esas posiciones porque fueron las que se les asignó.

**6.** Diseñen, construyan y prueben el método llamado ticTac() de la clase AManufacturing. Incluyan el caso siguiente en sus pruebas de unidad.



**7.** **a)** ¿Cómo quedarían simba y dala después de uno, dos y tres Tic-tac? Ejecuten el programa y hagan tres clics en el botón.

Ningún Tic-Tac: simba y dala están activos.



Un Ti-Tac: Siguen Activos.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Dos Tic-Tac: simba y dala están inactivos.

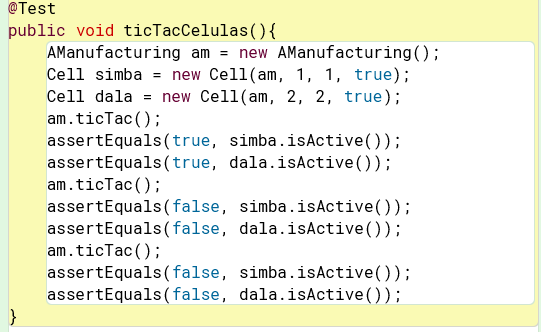
Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Tres Tic-Tac: Siguen inactivos.

Imagen que contiene biombo, edificio, verde, ventana

Descripción generada automáticamente



Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**b)** ¿Es correcto?

Sí, ya que el próximo estado depende de cuantos pasos ha dado, al hacer el primer Tic-Tac como ha dado 0 pasos seguirá activo (0 es múltiplo de 3), ya para el segundo como uno no es múltiplo de tres quedará en inactivo, al igual pasa con el tercer Tic-Tac dos no es múltiplo de tres, por lo tanto, siguen inactivos.

**Ciclo 2. Incluyendo a las células mimos**

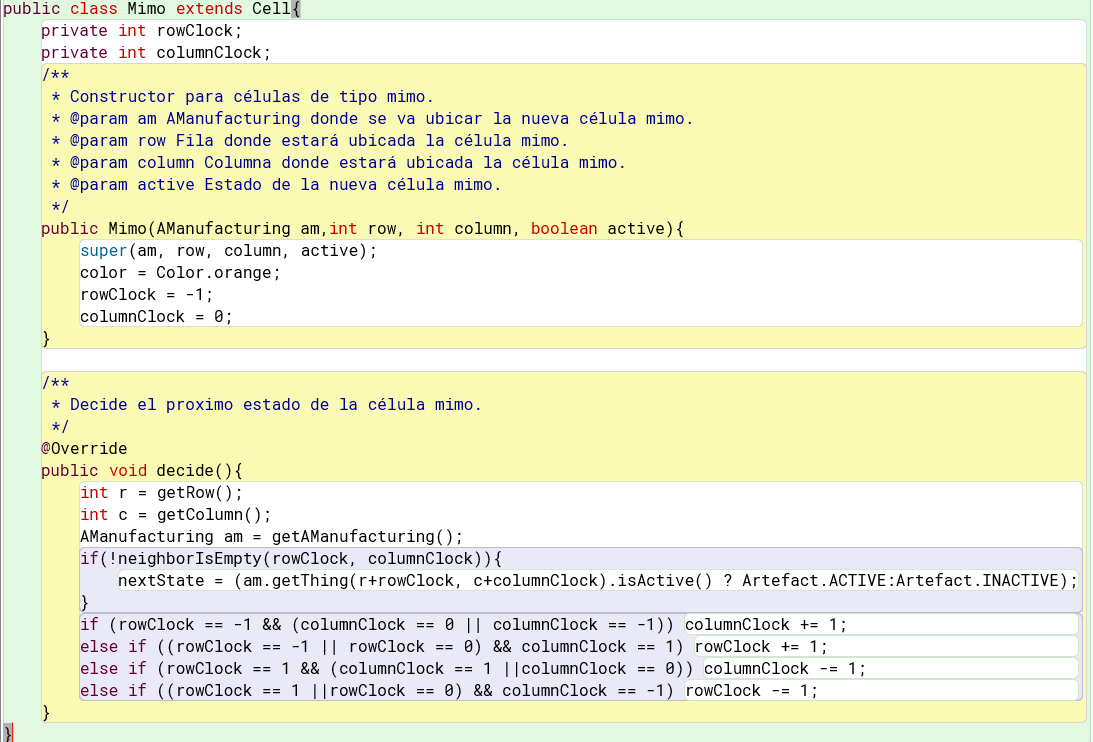
**1.** Para implementar esta nueva célula. ¿cuáles métodos se sobre-escriben (overriding)?

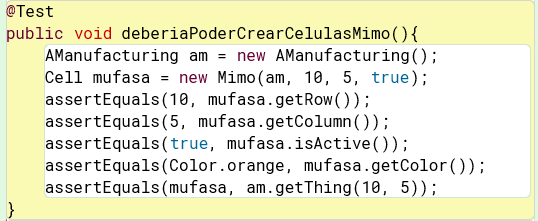
Se sobre-escribe el método *decide(),* ya que la decisión del estado de estas nuevas células es diferente al de una célula normal.

**2.** Diseñen, construyan y prueben esta nueva clase.

Escala de tiempo

Descripción generada automáticamente





Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**3.** Adicione un trio de mimos activos, llámelos mufasa, scar, rafiki. Coloquelos en línea horizontal.

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

Ningún Tic-Tac: mufasa, scar, rafiki están activos.

Imagen que contiene Tabla

Descripción generada automáticamente

Un Ti-Tac: Siguen Activos.

Imagen que contiene Tabla

Descripción generada automáticamente

Dos Tic-Tac: Siguen activos.

Imagen que contiene Tabla

Descripción generada automáticamente

Tres Tic-Tac: Siguen activos.

Imagen que contiene Tabla

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**b)** ¿Es correcto?

Sí, debido a que se le está dando como estado inicial activo, entonces al no encontrar ninguna célula para imitar al norte en el primer *Tic-Tac* seguirá con el estado que tenían, que es activo. En el segundo *Tic-Tac* tampoco encuentran ninguna célula (siguen activos) y por último el tercer *Tic-Tac* al oeste mufasa y scar imitan a scar y rafiki respectivamente que están en estado activo, así que siguen activos, por otro lado, rafiki no tienen a quien imitar, así que continua activo.

**Ciclo 3. Adicionando kriptonita**

**1.** Para incluir Kriptonita en AManufacturing.

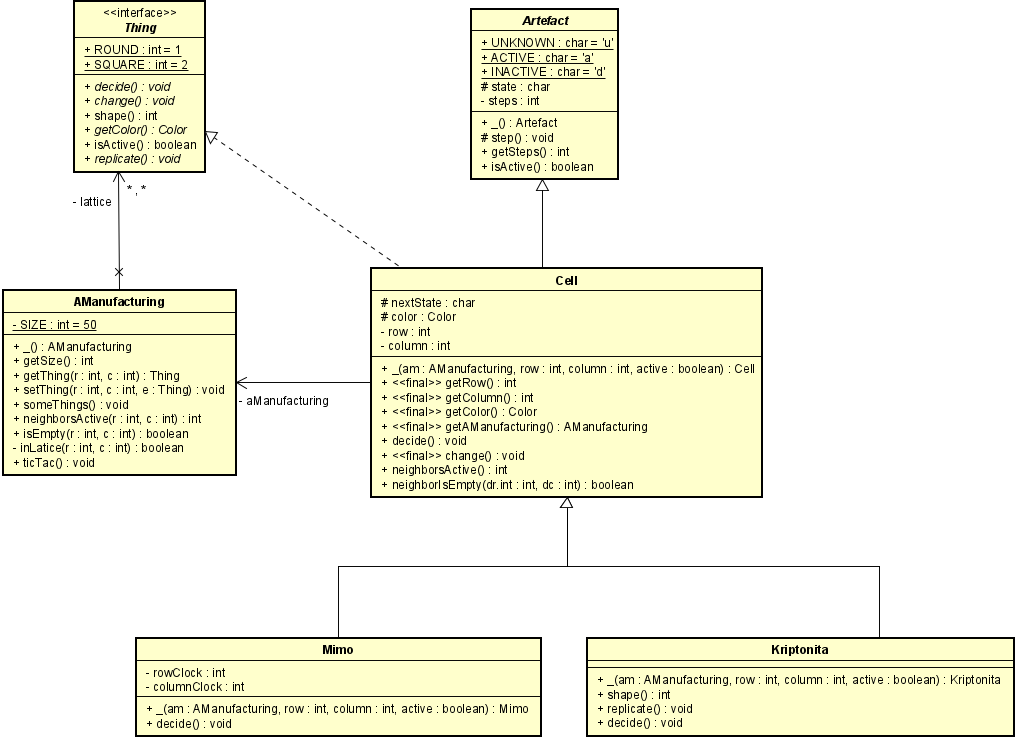
**a)** ¿debe cambiar en el código del AManufacturing en algo?

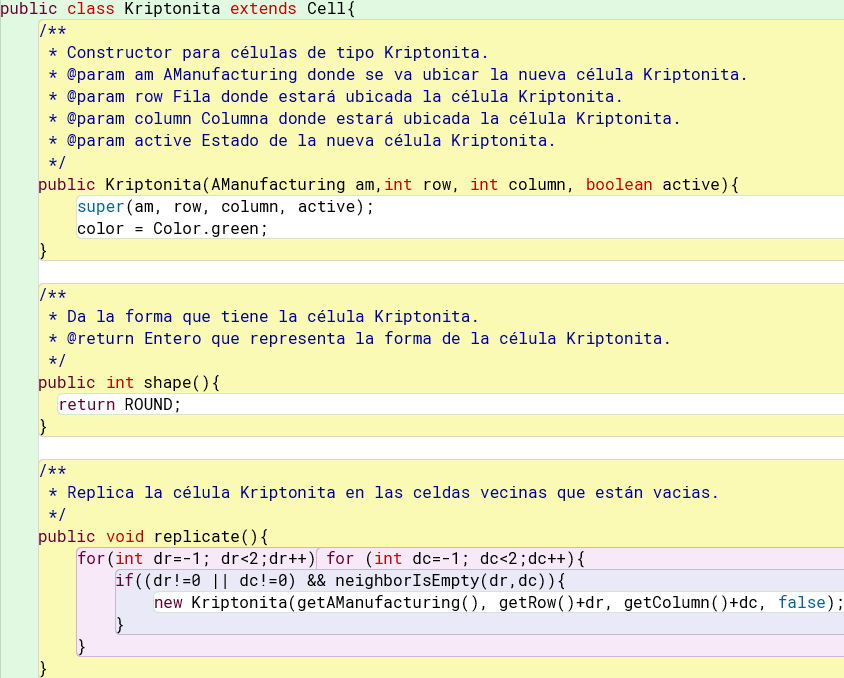
Debe cambiar el método *ticTac()*

**b)** ¿por qué?

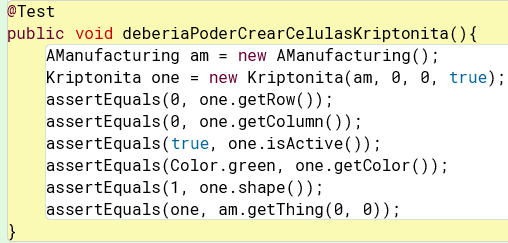
Porque ahora hay unas células que se replican, entonces por cada *Tic-Tac* se deben incluir las nuevas células.

**2.** Diseñen, construyan el método y prueben esta nueva clase.



 Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media



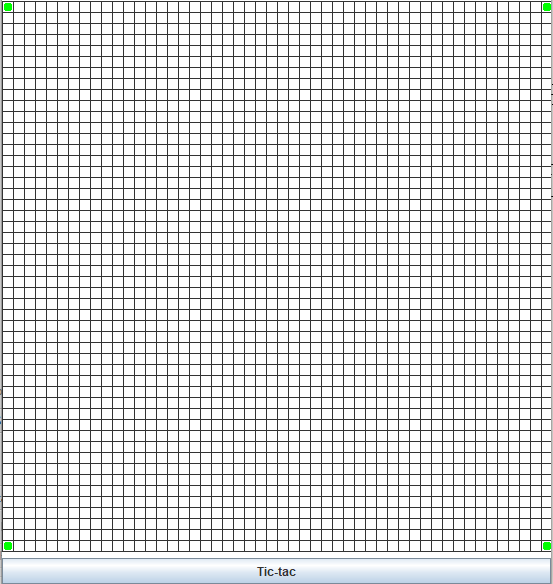
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

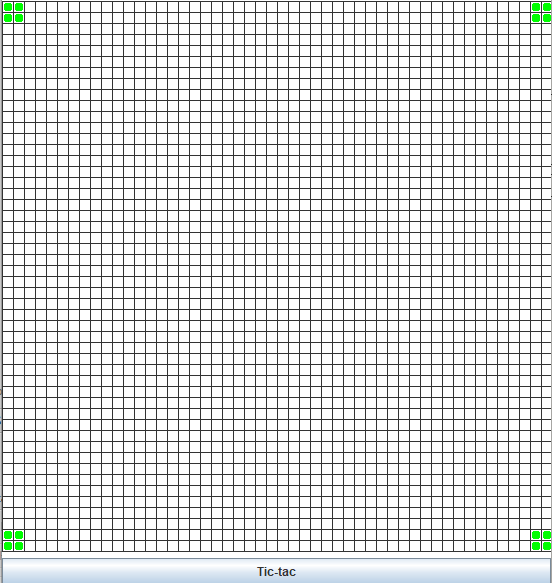
**3.** Adicionen kriptonita activa en las esquinas del AManufacturing, llámenlas one, two,three, four ,

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

Ningún Tic-Tac: Las células Kriptonita están activos.



Un Ti-Tac: Cada una se replica, crandose 12 nuevas céluas.



Dos Ti-Tac: Se replican las células creadas anteriormente, generando 20 células nuevas.

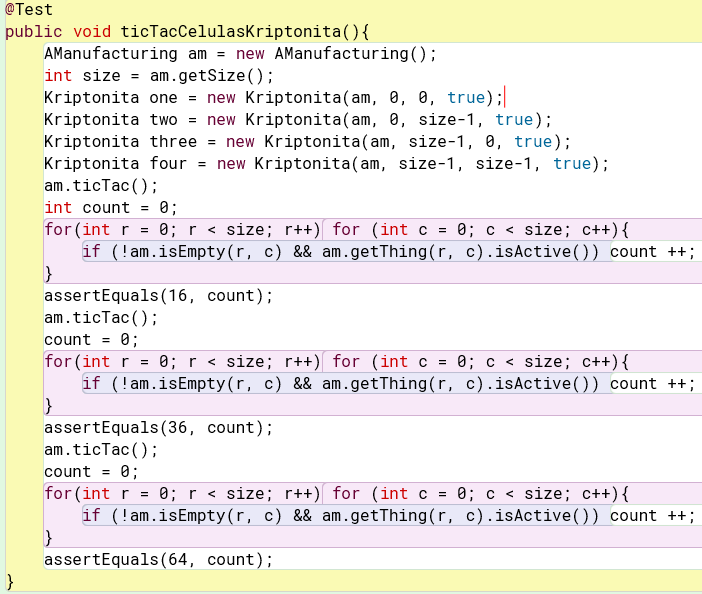
Imagen de la pantalla de una jaula

Descripción generada automáticamente con confianza baja

Tres Ti-Tac: Se replican las células creadas anteriormente, generando 36 células nuevas.

Imagen que contiene Tabla

Descripción generada automáticamente



Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**b)** ¿Es correcto?

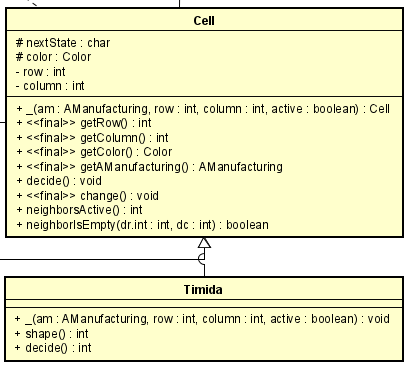
Sí, debido a que al hacer el primer *Tic-Tac* se replican las células en las celadas vecinas (se crean 3 por cada célula), por lo tanto, todas quedarían activas, ya que tendrían al menos un vecino. En el segundo *Tic-Tac* se vuelven reproducir generando 20 células nuevas en la red. Por último, el tercer *Tic-Tac* hace que se generen 36 células, lo que hace que la red quede con 64 células.

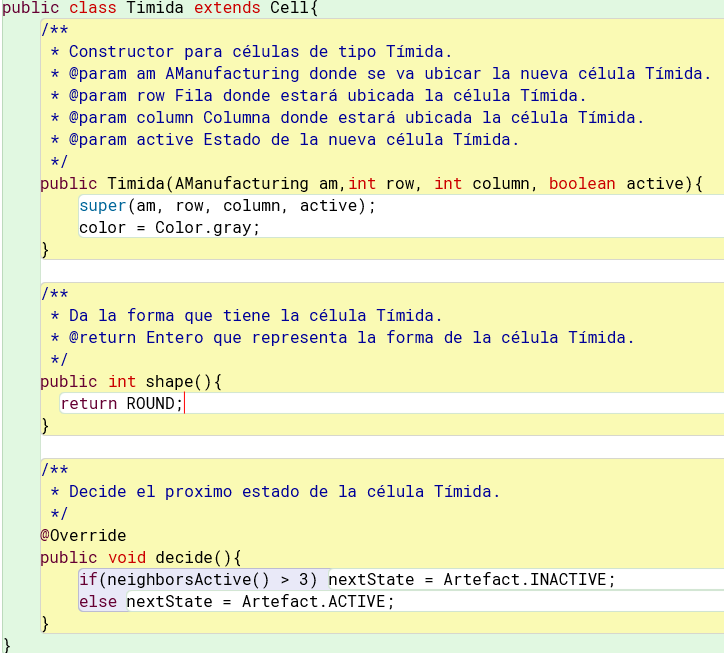
**Ciclo 4. Nueva Célula: Proponiendo y diseñando**

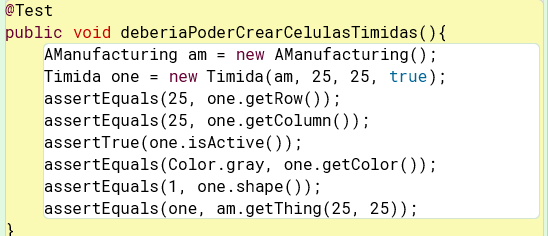
**1.** Propongan, describan e Implementen el nuevo tipo de células.

**Células tímidas:**

1. Son de color gris.
2. Son redondas.
3. Están activas cuando tiene menos de 4 vecinos activos, de lo contrario se desactivan.







Interfaz de usuario gráfica, Texto, Aplicación

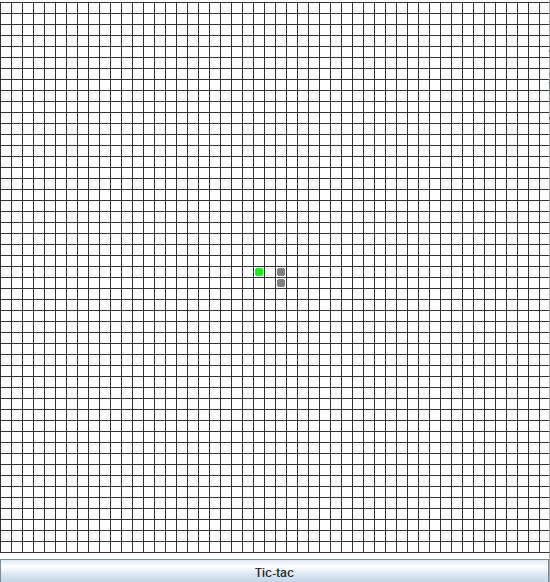
Descripción generada automáticamente

**2.** Incluyan una pareja de ellas con el nombre de ustedes. Piensen dos pruebas significativos y a2) expliquen la intención de cada caso. Incluyan las pruebas de unidad asociadas y ejecuten el programa para capturar las pantallas correspondientes.

**Prueba N°1:** Ubicar dos células tímidas activas en línea vertical una en (25, 25) y la otra en (24, 25), junto con una célula kriptonita activa en (24, 23). Después hacer tres *Tic-Tac.*

Está se prueba se hace para verificar que las células tímidas se desactiven cuando tienen alrededor más de 3 células activas.

Ningún Tic-Tac: Las células tímidas están activas.



Un Tic-Tac: Siguen activas.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Dos Tic-Tac: La ubicada en (24, 25) se desactiva

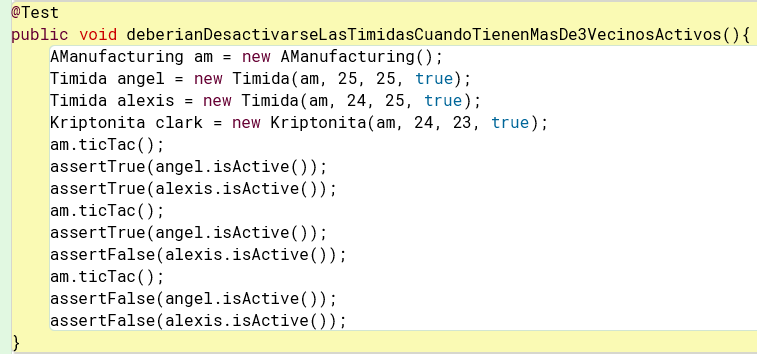
Imagen de la pantalla de una jaula

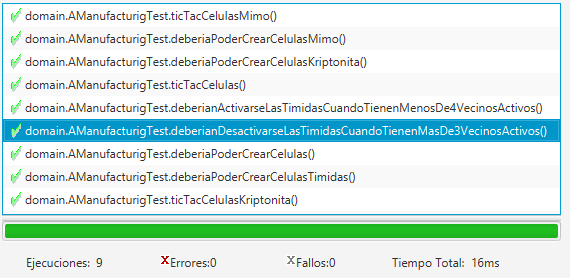
Descripción generada automáticamente con confianza baja

Tres Tic-Tac: Las dos están desactivadas.

Dibujo de una jaula

Descripción generada automáticamente con confianza baja





**Prueba N°2:** Ubicar dos células tímidas activas en línea vertical una en (25, 25) y la otra en (24, 25), junto con cinco células normales activas en (23, 25), (24, 24), (23, 24), (25, 26) y (26, 25). Después hacer tres *Tic-Tac.*

Está se prueba se hace para verificar que las células tímidas se activen cuando tienen alrededor menos de 4 células activas.

Ningún Tic-Tac: Las células están activas.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Un Tic-Tac: Las células se desactivan.

Imagen que contiene biombo, edificio, ventana, listo

Descripción generada automáticamente

Dos Tic-Tac: Se activa la célula ubicada en (25, 25).

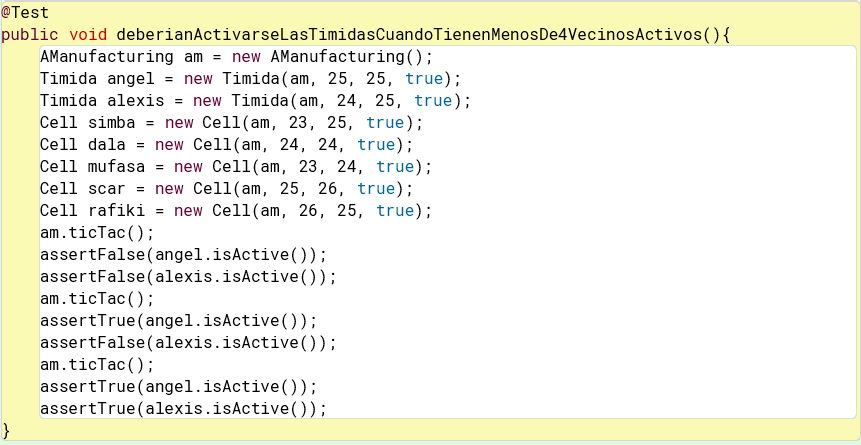
Imagen que contiene biombo, ventana, edificio, gente

Descripción generada automáticamente

Tres Tic-Tac: Las dos células están activadas.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja



Interfaz de usuario gráfica, Texto, Aplicación

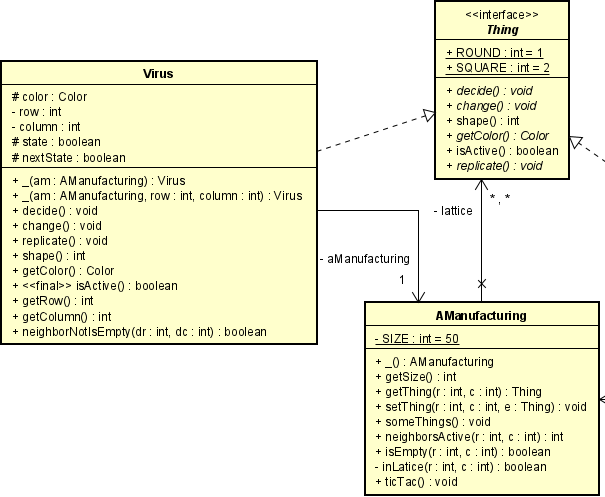
Descripción generada automáticamente

**Ciclo 5. Nuevas Cosas: Proponiendo y diseñando**

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

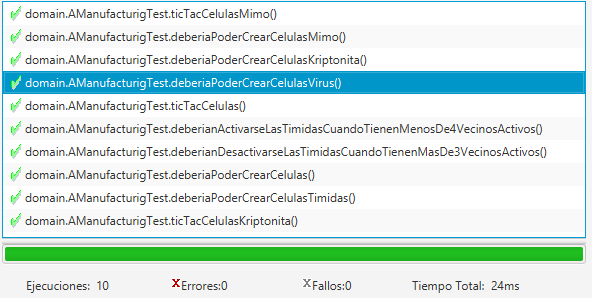
**Virus:**

1. Son de color rojo
2. Son redondas
3. Pueden generarse aleatoriamente o ser ubicadas dentro de la red bidimensional.
4. Siempre empiezan activas y se desactivan cuando todas las celdas vecinas tienen virus.
5. Solo pueden infectar a los elementos vecinos que estén activos.



Texto

Descripción generada automáticamente

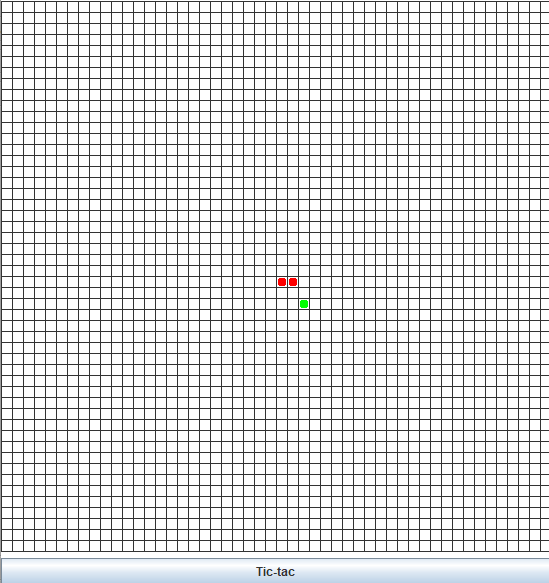


**2.** Incluyan un par de ellos con los nombres semánticos. Piensen dos pruebas significativos y a2) expliquen la intención de cada caso. Incluyan las pruebas de unidad asociadas y ejecuten el programa para capturar las pantallas correspondientes.

**Prueba N°1:** Ubicar dos virus en línea horizontal uno en (25, 25) y el otro en (25, 26), junto con una célula kriptonita activa en (27, 27). Después hacer tres *Tic-Tac.*

Está se prueba se hace para verificar que los virus se desactiven cuando todas las celdas vecinas estén con virus.

Ningún Tic-Tac: Los dos únicos virus están activos.



Un Tic-Tac: Siguen activos y ahora tiene dos elementos vecinos.

Imagen que contiene biombo, edificio, ventana, gente

Descripción generada automáticamente

Dos Tic-Tac: Infecta a los dos vecinos que tenían, ahora hay cuatro] virus.

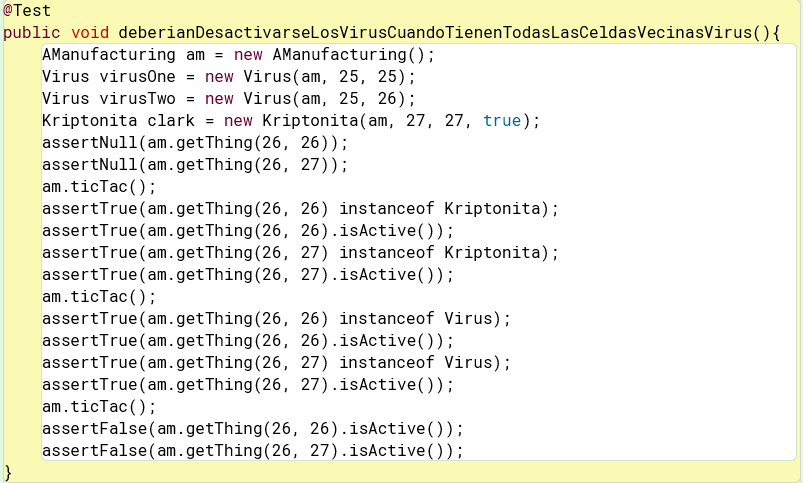
Imagen que contiene biombo, edificio, gente, ventana

Descripción generada automáticamente

Tres Tic-Tac: Dos virus se desactiva al tener en todas las celdas vecinas virus.

Imagen que contiene biombo, edificio, verde, jaula

Descripción generada automáticamente



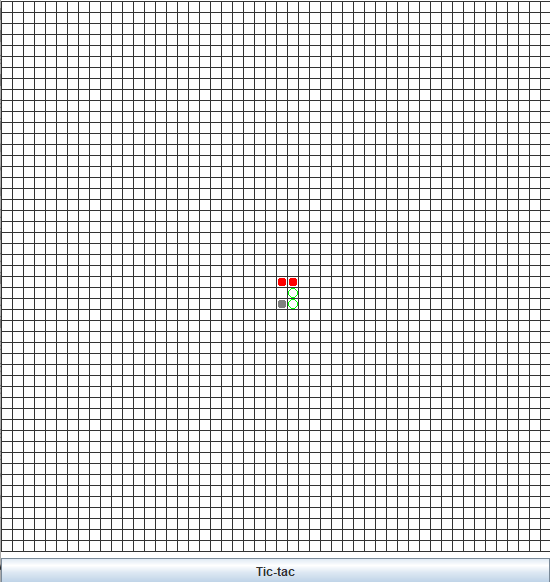
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

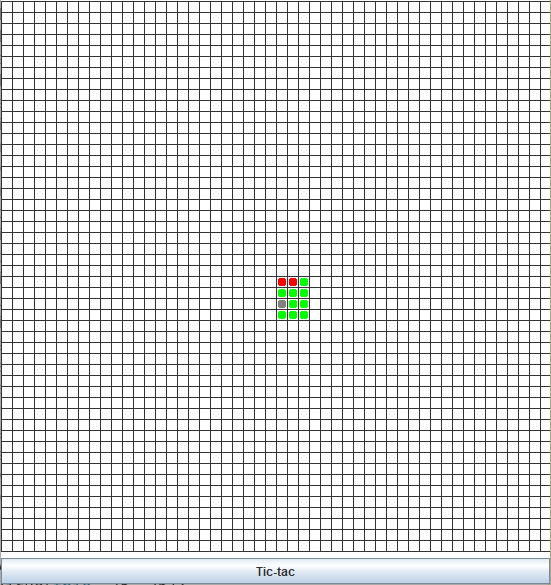
**Prueba N°2:** Ubicar dos virus en línea horizontal en (25, 25) y el otro en (25, 26), junto con dos células kriptonita inactivas en línea vertical en (26, 26) y (27, 26) y una célula tímida activa en (27, 25). Después hacer tres *Tic-Tac.*

Está se prueba se hace para verificar que los virus solo infecten a elementos vecinos que están activos.

Ningún Tic-Tac: Solo hay dos virus.



Un Tic-Tac: Siguen solo los dos virus. No infectan a la Kriptonita que estaba descativada.



Dos Tic-Tac: Infectan a sus vecinos y la célula tímida se desactiva.

Imagen que contiene biombo, edificio, gente, equipaje

Descripción generada automáticamente

Tres Tic-Tac: La célula tímida no se infecta.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Caso 6. Replicate con Rfplicbte**

**1.** Diseñen, construyan y prueben esta nueva clase.

Diagrama

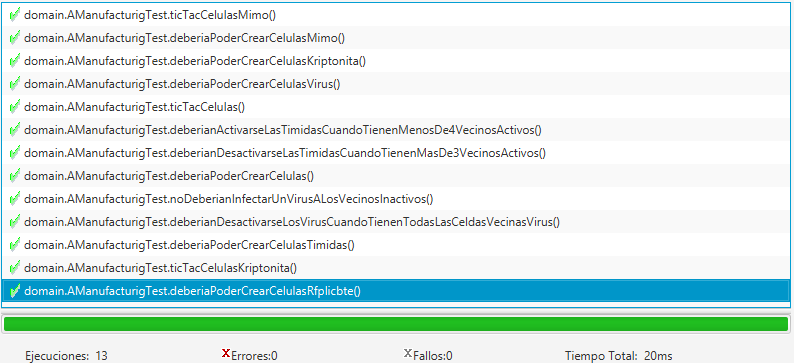
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente



**2.** Adicionen el caso de la figura inicial y de los cuatro Tic-tac, ¿qué pasa? ¿funciona? Capture las cuatro pantallas del ejemplo.

Ningún Tic-Tac

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

Un Tic-Tac

Imagen que contiene biombo, edificio, ventana, gente

Descripción generada automáticamente

Dos Tic-Tac

Dibujo de una jaula

Descripción generada automáticamente con confianza baja

Tres Tic-Tac

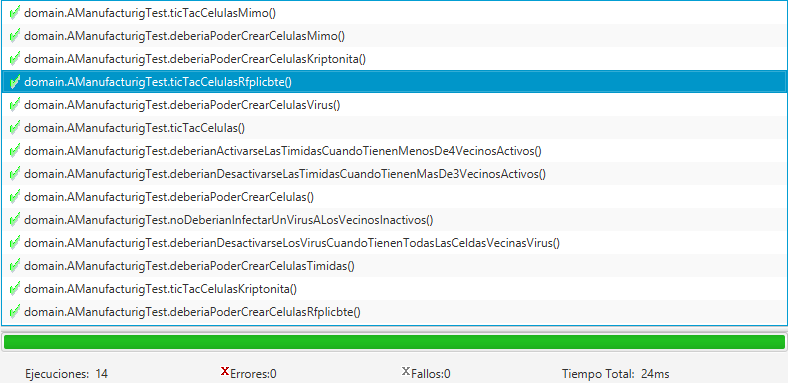
Imagen que contiene biombo, edificio, maleta, jaula

Descripción generada automáticamente

Cuatro Tic-Tac

Dibujo de una jaula

Descripción generada automáticamente con confianza baja



**Empaquetando la versión final para el usuario.**

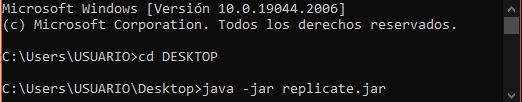
**1.** Revise las opciones de BlueJ para empaquetar su programa entregable en un archivo .jar. Genere el archivo correspondiente.

Se abre el menú del proyecto en *BlueJ* y se crea el archivo *.jar* eligiendo la clase principal, que en este caso sería *AManufacturingGUI.*



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

**Comando:** *java -jar <Nombre del archivo>.jar*



Al ejecutarlo abre la interfaz gráfica del programa.

Un dibujo de una jaula

Descripción generada automáticamente con confianza baja

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

* Genera seguridad para el programador, ya que se entrega de forma comprimida en un solo archivo y no se ve el código fuente.
* Fácil portabilidad, debido a que lo único que se necesita es JVM (Java Virtual Machine).
* Ocupa menos espacio porque esta comprimido.

**DE BLUEJ A CONSOLA**

**Comandos básicos del sistema operativo**

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

* **Crear:** *md <NombreDirectorio>*
* **Borrar:** *rd <NombreDirectorio/NombreArchivo>*
* **Listar:**

*dir*

*tree* <*NombreDirectorio*> (lista completa del árbol de directorios)

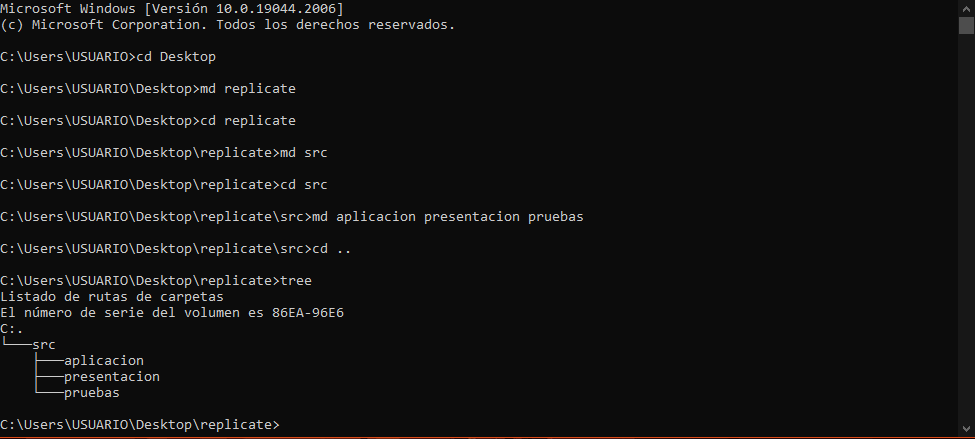
* **Copiar:**

*copy* <NombreArchivo> <NombreDirectorioDestino>

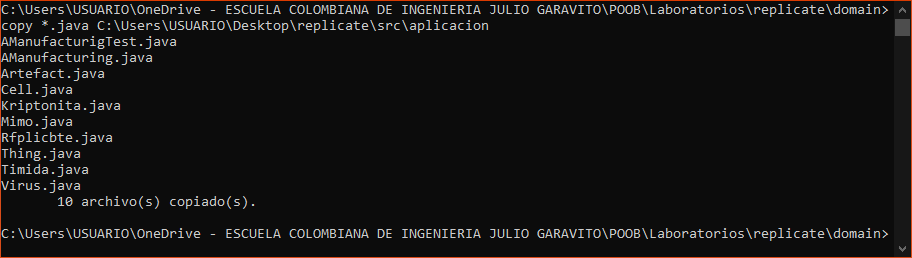
*xcopy* <NombreDirectorioOrigen> <NombreDirectorioDestino> (Copia todos los archivos de un directorio a otro)

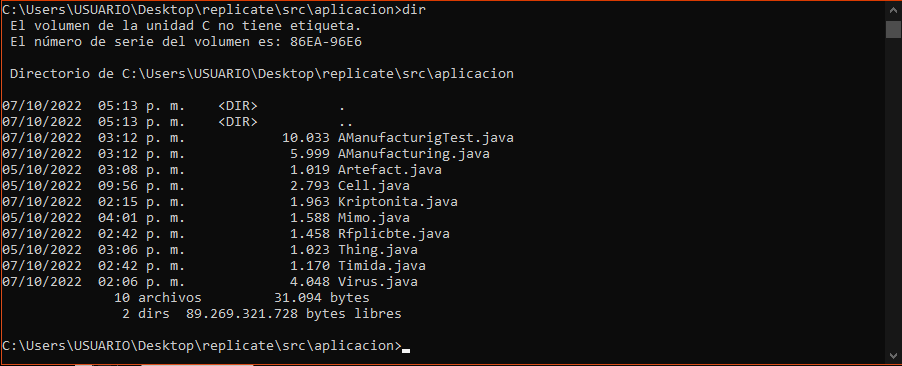
* **Eliminar:** *del <NombreDirectorio/NombreArchivo >* (Elimina el archivo o archivos de la carpeta, más no la carpeta)

**2.** 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.



**3**. En el directorio copien únicamente los archivos \*.java del paquete de aplicación . Consulte y capture el contenido de src/aplicación





**Estructura de proyectos java**

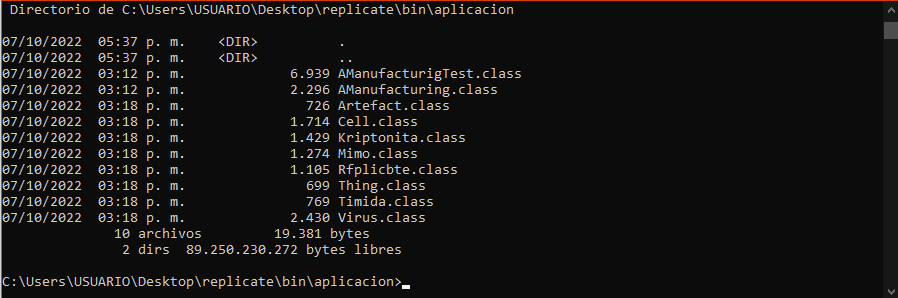
**1.** Investiguen los archivos que deben quedar en cada una de esas carpetas y la organización interna de cada una de ellas.

* **src:** Deben estar todos los códigos fuente *.java.*
* **bin:** Deben estar los *bytecodes* de los códigos fuentes *.class.*
* **docs:** Debe estar la documentación.

La organización interna de los anteriores va a ser por los paquetes definidos dentro del proyecto.

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

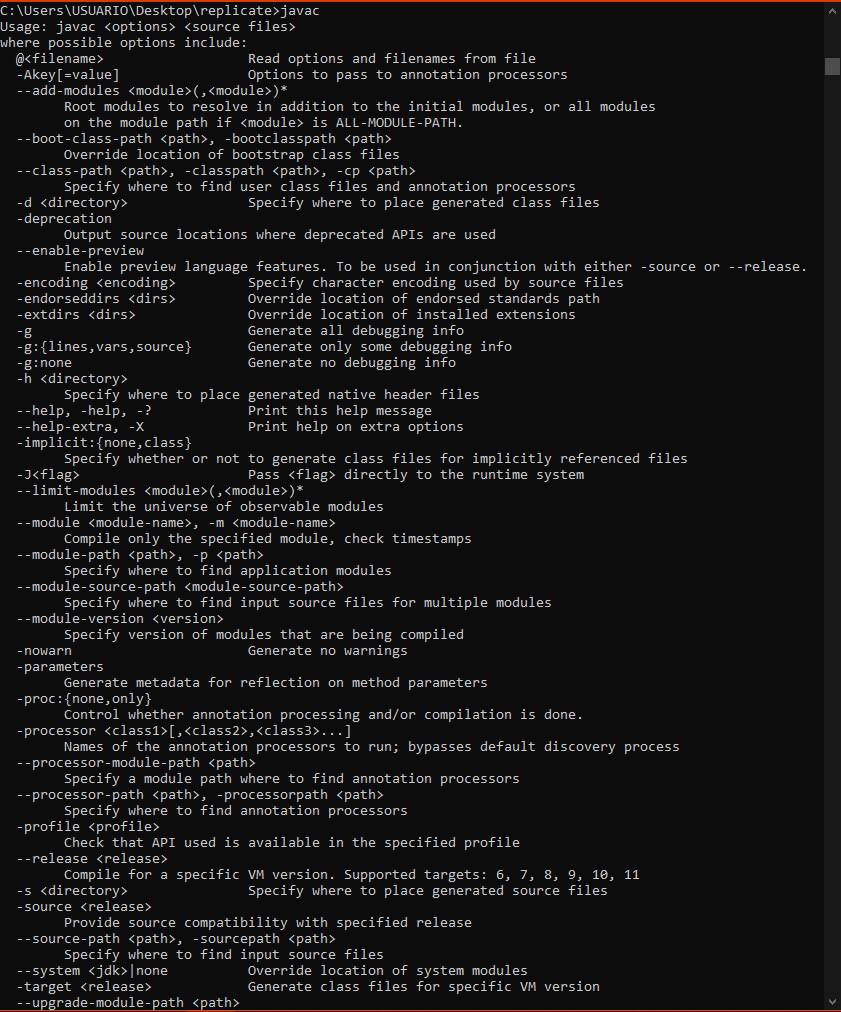
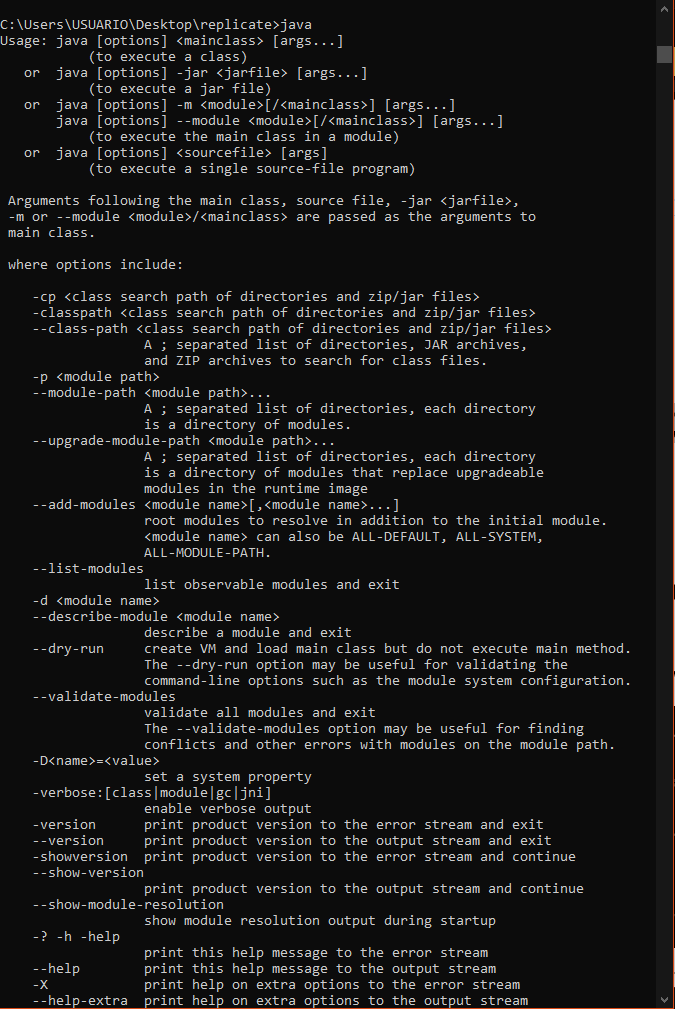
Se debería copiar los archivos *.class* porque son los que con tienen el *bytecode* del proyecto y son los que interpretados y ejecutados por la máquina virtual java.

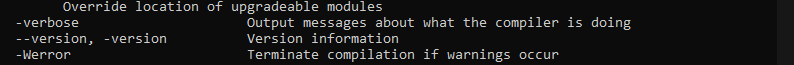
****

**Comandos de java**

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

* **javac:** Comando que permite al sistema operativo copilar código fuente de Java.
* **java:** Llama al intérprete de java que ejecuta el *bytecode* generado por javac.
* **javadoc:** Genera la documentación API del código fuente.
* **jar:** Comprime proyectos java en un solo archivo .jar y también sirve para ejecutarlos.

**2.** Cree una sesión de consola y consulte en línea las opciones de los comandos java y javac. Capture las pantallas.

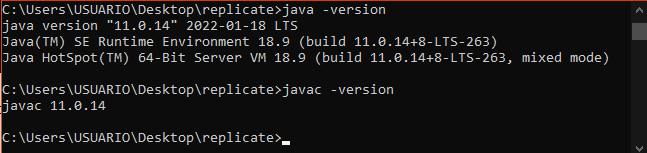


Texto

Descripción generada automáticamente

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

Para saber la versión es: *<comando> -version*



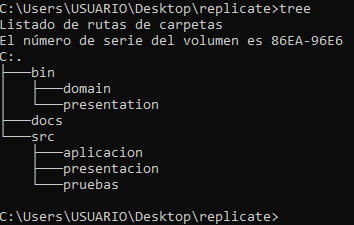
**Compilando**

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

**Comando:** *javac -d bin src\aplicacion\\*.java src\presentacion\\*.java*



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

 Texto

Descripción generada automáticamenteTexto

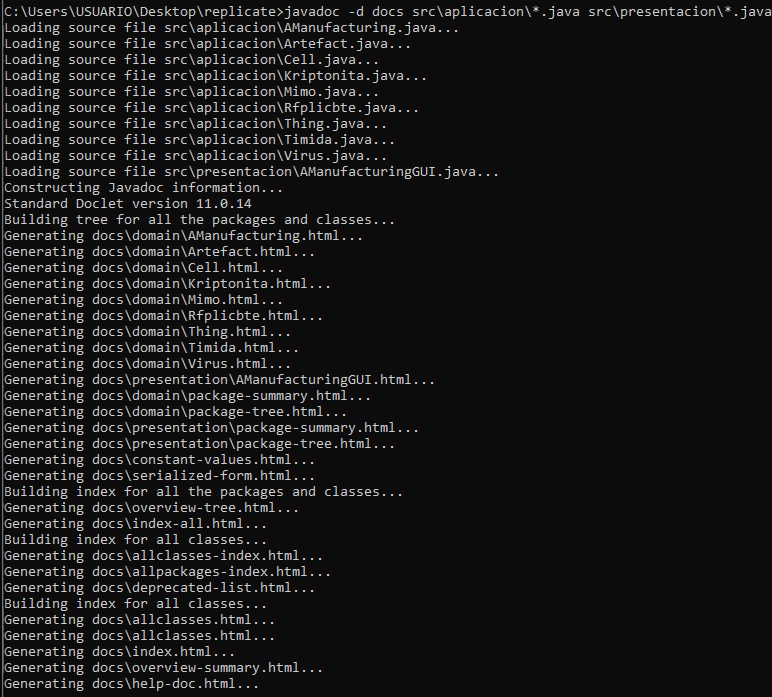
Descripción generada automáticamente

Aparecen los archivos *.class* con el *bytecode* del proyecto separados en directorios con los nombres de los paquetes.

**Documentando**

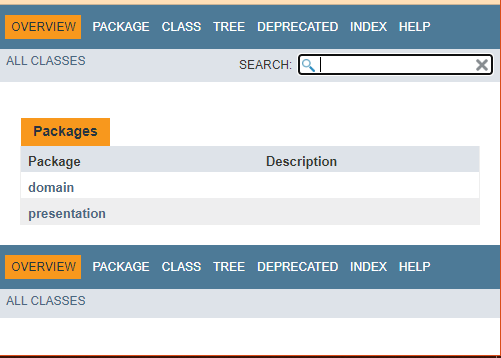
**1.** Utilizando el comando javadoc, desde el directorio raiz, genere la documentación (API) en formato html, en este directorio. ¿cuál es el comando completo para generar esta documentación?

**Comando:** *javadoc -d docs src\aplicacion\\*.java src\presentacion\\*.java*



**2.** ¿Cuál archivo hay que abrir para empezar a navegar por la documentación? Ábralo y capture la pantalla.

Para navegar por la documentación hay que abrir el archivo *index.html*

**

**Ejecutando**

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

**Comando:** *java -cp bin presentation.AManufacturingGUI*

**

**Como utilizarlo:**

*java -cp /path/to/bin <NombrePaquete>.<NombreClasePrincipal.*

**Probando**

**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. ¿Cómo se incluye un paquete para compilar? ¿Qué instrucción completa tuvo que dar a la consola para compilar?

Para poder copilar las pruebas tocó descargar *junit-4.13.2.jar* y *hamcrest-core-1.3.jar* y se colocaron en el directorio raíz. Además, se editó el archivo *.java* de las pruebas, cambiando las importaciones org.junit.jupiter.api.\* por org.junit.\*.

Si se quiere incluir un nuevo paquete toca poner la dirección que tiene en este caso era *src\pruebas\\*.java*

Para completar la instrucción se agregó ***-cp*** *junit-4.13.2.jar* junto con el paquete de pruebas.

**Comando**: *javac -d bin -cp junit-4.13.2.jar src\aplicacion\\*.java src\presentacion\\*.java src\pruebas\\*.java*

**

**2.** Ejecute desde consola las pruebas . ¿Cómo utilizó este comando?. Puede ver ejemplos de cómo ejecutar el“test runner”en: <http://junit.sourceforge.net/doc/cookbook/cookbook.htm>

**Comando**: *java -cp junit-4.13.2.jar;hamcrest-core-1.3.jar;bin org.junit.runner.JUnitCore domain.AManufacturigTest*

**3.** Pegue en su documento el resultado de las pruebas

Imagen que contiene Texto

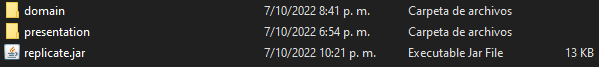
Descripción generada automáticamente

**Empaquetando**

**1.** Consulte como 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). ¿Cómo empaquetó jar ?

**Comando:** *jar cfe replicate.jar presentation.AManufacturingGUI domain\\*.class presentation\\*.class*





**2.** ¿Cómo se ejecuta el proyecto empaquetado?

Se ejecuta dando doble click sobre el archivo *.jar* o desde consola con el comando*: java -jar replicate.jar*



**RETROSPECTIVA**

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

Hombre)

Por cada integrante 20 Horas.

**2.** ¿Cuál es el estado actual de laboratorio? ¿Por qué? (Para cada método incluya su

estado)

Realizado completamente.

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

importante?

Porque permite que el código nuevo se libere al repositorio de código fuente por turnos, lo que ayuda a identificar de manera consistente una versión mas reciente.

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

problema? ¿Qué hicieron para resolverlo?

El mayor logro fue hacer la célula Rfplicbte, porque al principio no comprendíamos bien el enunciado y su implementación nos costó.

El mayor problema fue entender claramente el funcionamiento de las clases abstractas y las interfaces, para solucionarlo investigamos más.

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

resultados?

Elaborar el laboratorio con anticipación. No comprometemos a realizar más conjuntamente el laboratorio.