

PROGRAMACIÓN ORIENTADA A OBJETOS

Herencia e interfaces ADEMÁS Java

desde consola 2020-I

Laboratorio 3/6

Contexto

Un *autómata celular* (A.C.)¹ es un *modelo matemático* para *representar sistemas* que puedan ser *descritos* como una *colección masiva* de *objetos simples* que *interactúan localmente* unos con otros y que *evolucionan a pasos discretos*. Sus *características* son:

1. Las *células* se ubican en una *rejilla*, máximo una en cada *celda*.
2. Una *célula* puede estar en uno de un *conjunto posible* de *estados*. En nuestro caso: *viva (•)* o *muerta (+)*.
3. Cada *célula* decide qué va *suceder* en la *siguiente etapa* de *tiempo* de acuerdo a su *naturaleza*, su *estado* y sus *vecinas*.
4. En cada *momento*, todas las *células* toman la *decisión* de su *acción futura* y luego todas *cambian*. Si hay *nacimiento* en su *decisión*, este será *real al instante siguiente*.

Conociendo

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

a) ¿Cuántos paquetes tiene?

- Tiene dos: presentación y aplicación

b) ¿Cuántas clases tiene en total?

- Tiene cuatro:
 - AutomataCelular
 - Celula
 - Elemento
 - AtomataGUI

¿Cuántas tienen fuentes?

Todas tienen fuente la cual se evidencia en los archivos .java

c) ¿Cuál es la clase ejecutiva? ¿Por qué?

AutomataCelular, porque es la clase de inicialización del programa.

2. Ejecuten el programa.

¿Qué funcionalidades ofrece?

Se puede crear una instancia de AutomataCelular, donde ofrece:

- Obtiene la longitud
- Obtiene el elemento
- Se puede asignar un elemento en la matriz
- Algunos elementos
- Tictac

¿Qué hace actualmente? ¿Por qué?

Genera una cuadrícula vacía

Arquitectura general.

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

- **Package:** se usa para agrupar clases, su comportamiento es similar a la de un directorio, se utiliza para no recurrir en conflictos de nombre, escribir mejor y hacer un código sostenible.
- **Import:** se utiliza para importar y usar paquetes de java o creados por el usuario.

2. *Revise el contenido del directorio de trabajo y sus subdirectorios. Describa su contenido. ¿Qué coincidencia hay entre paquetes y directorios?*
 - Directorios de trabajo:
 - Aplicación:
 - Clase AutomataCelular
 - Clase Celula
 - Clase Elemento
 - Presentación:
 - AutomataGUI
 - Doc:
 - Se encuentra la documentación del proyecto.
3. *Inicie el diseño con un diagrama de paquetes en el que se presente los componentes y las relaciones entre ellos.*

Arquitectura detallada.

1. *Usando ingeniería reversa preparen el proyecto para **MDD**. Presente el diseño estructural actual de la aplicación (diagrama de clases). Las clases de la capa de presentación sólo deben tener los elementos públicos.*
2. *Adicione en las fuentes la clase de pruebas necesaria para **BDD**. (No lo adicione al diagrama de clases)*
 - *¿En qué paquete debe estar? ¿Por qué?*
 - Debe estar en el paquete de aplicación ya que este se encarga de definir la funcionalidad del programa.
 - *¿Asociado a qué clase? ¿Por qué?*
 - AutomataCelular debido a que esta es la clase principal del paquete y este utiliza las demás clases.

Ciclo 1. Iniciando con las células normales

1. **Estudie la clase AutomataCelular**
 - *¿Qué tipo de colección usa para albergar los elementos?*
 - Utiliza un array de dos dimensiones.
 - *¿Puede recibir células? ¿Por qué?*
 - Si, por medio del método setElemento, se recibe un parámetro de tipo elemento (interfaz), el cual es implementado por la clase célula.
2. **Estudie el código de la clase Celula, ¿qué otras clases la definen? ¿cómo?**
 - La clase Célula implementa la clase Elemento, es decir, implementa todos sus métodos abstractos.
3. **Todas las células, ¿qué saben hacer?**
 - Retornan sus propiedades como lo son: fila, columna, color, si está viva, edad.
 - Decide cuál será su siguiente estado.
 - Cambia su estado.

¿qué no puede hacer distinto?

 - Retornar la fila, la columna, color y si está viva la célula

¿qué debe aprender a hacer? Justifique su respuesta.

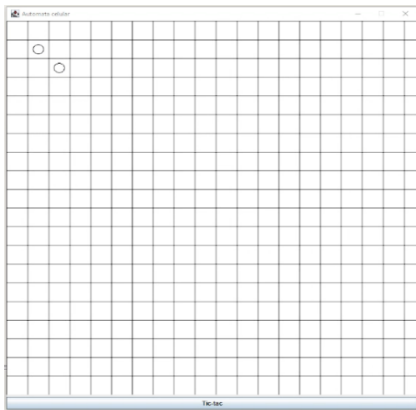
 - La célula debe aprender a decidir y a cambiar.
4. **Por comportarse como un elemento,**
 - *¿qué sabe hacer?*
 - Obtener la forma y si está viva.

- *¿qué no puede hacer distinto?*
 - getColor, ya que es un método abstracto y no es posible para la clase implementarlo.
- *¿qué debe aprender a hacer? Justifique su respuesta.*
 - Decida y cambie, ya que los métodos no tienen su cuerpo implementado.

5. Considerando lo anterior, una Celula:

- *¿de qué color es?*
 - Es color negro, ya que en el constructor de la célula se define así.
- *¿cómo decide?*
 - Una célula de dos o más años de edad, muere.
- *¿cómo cambia? Justifiquen sus respuestas.*
 - Su edad aumenta un año, y su estado actual pasa a ser el siguiente.
- *Ahora vamos a crear dos células en diferentes posiciones (1,1) (2,2) llámelos indiana y 007 usando el método algunosElementos(). Ejecuten el programa, ¿Cómo quedan todas las células? Capturen una pantalla significativa.*

```
public void algunosElementos(){
    automata[1][1]=new Celula(this,1,1,"indiana");
    automata[2][2]=new Celula(this,2,2,"007");
}
```



6. En este punto vamos a construir (diseño y código) el método que atiende el botón Tic-tac: el método llamado ticTac() de la clase AutomataCelular.

¿Cómo quedarían indiana y 007 después de uno, dos y tres Tic-tac?

- Un Tic-Tac: Las células están negras, lo que significa que nacieron.
- Dos Tic-Tac: Las células se mantienen vivas.
- Tres Tic-Tac: Las células mueren.

Escriba la prueba correspondiente.

7. Construyan el método. ¿Es correcto?

- Si.

8. Ejecuten el programa y hagan tres clics en el botón. ¿Cómo quedan las células? Capturen una pantalla significativa.

Primero clic:



Ciclo 2. Incluyendo a las células izquierdosas

El objetivo de este punto es incluir en el autómatas algunas células “izquierdosas”. Estas células son de color rojo y deciden morir si hay algún elemento vivo a su derecha (oeste).

1. Si tenemos seguidas dos células izquierdosas vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué?

- En el primer click las dos células nacen, en el segundo click la célula de la izquierda muere y en el tercer click el tablero permanece igual.

2. ¿Cuáles son las adiciones necesarias en el diseño?

- Adicionar la clase Izquierdosa con su constructor.

¿y los cambios? ¡Hágalos!

- Adicionar la herencia de la clase Izquierdosa a la clase Celula.

¿cuáles métodos se sobre-escriben (overriding)?

- Decida, ya que Izquierdosa se comporta de forma diferente a Célula cuando decide su estado siguiente.

Ahora escriba el código correspondiente a la célula Izquierdosa ¿Las pruebas son correctas?

- Si, las pruebas fueron correctas.

3. Para aceptar la célula Izquierdosa en AutomataCelular, ¿debe cambiar el código del AutomataCelular en algo? ¿por qué?

- No, debido a que extiende a la clase Celula, por tanto, es una Celula y la clase AutomataCelular la trata como tal.

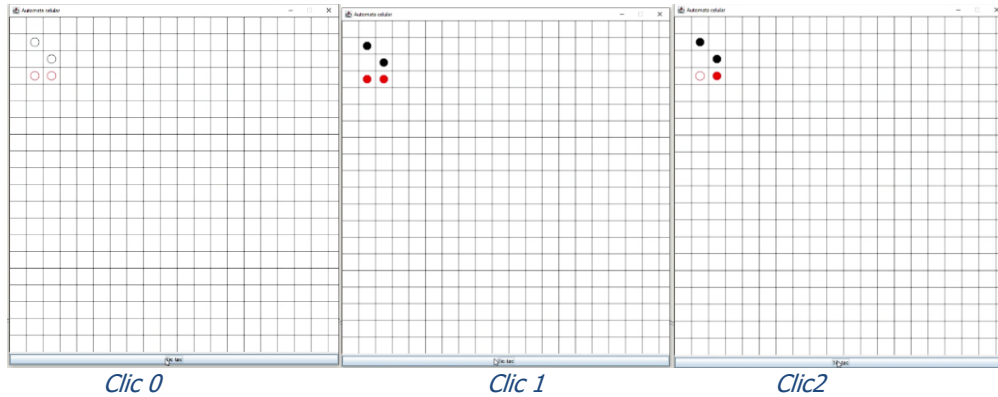
4. Adicionen juntas una pareja de células izquierdosas en la fila 3, llámenlas marx y hegel, ¿Cómo quedarían después de uno, dos y tres **Tic-tac**? Escriba la prueba correspondiente.

- Un Tic-Tac: Ambas células están vivas.
- Dos Tic-Tac: La célula de la izquierda muere.
- Tres Tic-Tac: El tablero sigue igual.

5. Construyan el método. ¿Es correcto?

- Sí, es correcto.

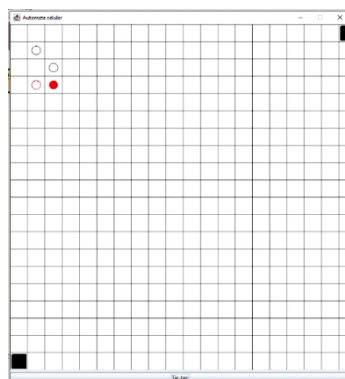
6. Ejecuten el programa y hagan dos clics en el botón. ¿Cómo quedan las células? Capturen una pantalla significativa.



Ciclo 3. Adicionando una barrera

El objetivo de este punto es incluir en el AutomataCelular barreras (sólo vamos a permitir un tipo de barreras). Las barreras son cuadradas, negras y como barreras están muertas: ni deciden ni cambian.

1. Construyan la clase Barrera para poder adicionarla en el AutomataCelular ¿qué hicieron?
 - En la interfaz Elemento, se cambió el método getForma de tal forma que sea abstracto, ya que ahora no todos los elementos son de una sola forma.
 - En la clase Celula, implementamos el método getForma.
 - En la clase Barrera, implementamos los métodos: getForma, getColor, getFila, getColumna, getNombre, isVivo.
2. Para aceptar este elemento, ¿debe cambiar el código del AutomataCelular en algo? ¿por qué?
 - No porque las referencias de los elementos que están en el AutomataCelular son de tipo Elemento, y nuestra clase Barrera implementa la interfaz Elemento, por tanto, esta referencia sigue siendo válida.
3. Adicionen dos Barreras cerca en las esquinas del AutomataCelular, llámenlas suroeste y noreste, ¿Cómo quedarían después de uno, dos y tres Tic-tac? Escriba la prueba correspondiente.
 - Las barreras deberían ser estáticas sin importar el número de Tic-Tacs aplicados.
4. Construyan el método. ¿Es correcto?
 - Sí, es correcto.
5. Ejecuten el programa y hagan tres clics en el botón. Capturen una pantalla significativa. ¿Qué pasa? ¿es correcto?
 - Las barreras permanecen muertas, el comportamiento es correcto.



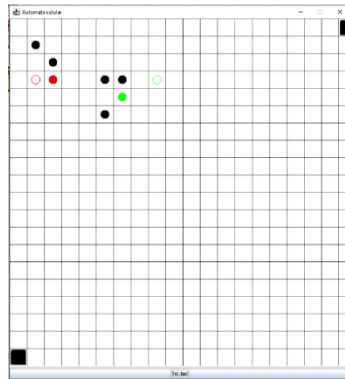
Ciclo 4. Nueva Célula: Proponiendo y diseñando

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

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

Implementaremos un nuevo tipo de Célula llamada Sociable, la cual es de color verde y revive si tiene tres o más células vivas a su alrededor, incluyendo las diagonales, o muere si no se cumple esta condición.

2. *Incluyan una pareja de ellos con el nombre de ustedes. ejecuten el programa con dos casos significativos. Explique la intención de cada caso y Capturen las pantallas correspondientes.*



La célula Sociable que está rodeada por otras tres células está viva, mientras que la que está sola, está muerta.

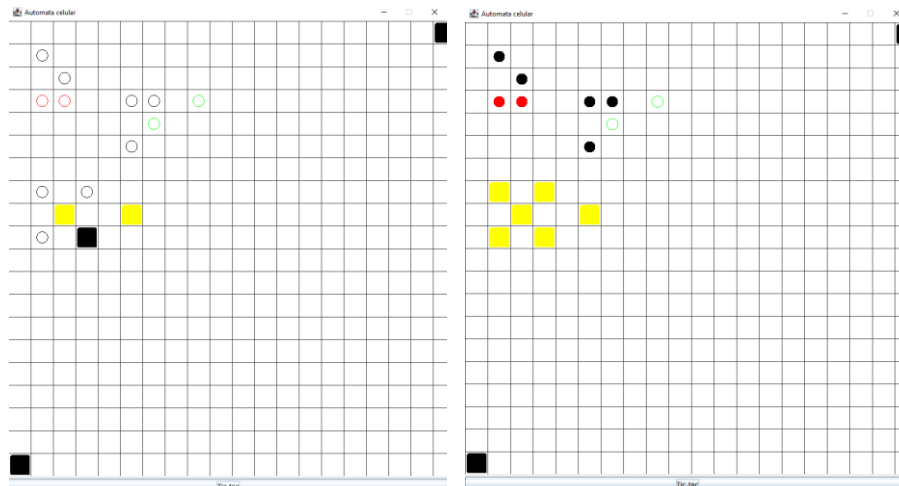
Ciclo 5. Nuevo elemento: Proponiendo y diseñando

El objetivo de este punto es permitir recibir en un nuevo elemento (no célula) en el Automata Celular.

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

Se desea agregar un bloque de forma cuadrada y de color amarillo que se denominará bloque infeccioso, el cual toma todas las células y bloques que están a su alrededor y las transforma en copias de el mismo. El bloque está muerto, es un misterio como realiza la infección.

2. *Incluyan un par de ellos con nombres semánticos. ejecuten el programa con dos casos significativos. Explique la intención de cada caso y Capturen las pantallas correspondientes.*



El infeccioso de la derecha no tiene a quien infectar, sin embargo, el de la izquierda infecta a todos los elementos a su alrededor, incluyendo las barreras

Caso 6. El Juego de la vida

El juego de la vida es el mejor ejemplo de un autómata celular, diseñado por el matemático británico John Horton Conway en 1970. Un Automata Celular es una malla con células. Las células pueden estar vivas o muertas y pueden estar listas para vivir o para morir en el siguiente momento. Cada célula tiene como vecinas las que están próximas a ella, incluso en las diagonales.

En el juego de la vida el estado del Automata Celular evoluciona a lo largo de unidades de tiempo y los cambios dependen del número de células vecinas vivas:

- Una célula muerta con exactamente 3 células vecinas vivas "revive" (al tiempo siguiente estará viva).
- Una célula viva con 2 ó 3 células vecinas vivas sigue viva.
- Si la célula tiene una o más de tres vecinas muere o permanece muerta por "soledad" o superpoblación".
- Si en el vecindario, hay una celda vacía rodeada por 3 células vivas "nace" una nueva célula (al tiempo siguiente estará viva).

Primero todas las células toman la decisión de lo que pasará en el tiempo siguiente y luego la realizan.

Existen numerosos tipos de patrones que pueden tener lugar en el juego de la vida:

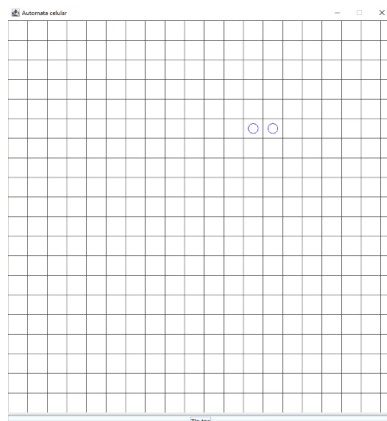
El bloque y el barco son estáticos, el parpadeador y el sapo son osciladores y el planeador y la nave espacial ligera viajan por el Automata Celular.



1. Si tenemos seguidas dos células Conway vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriba la prueba correspondiente.
 - Primer clic: las células nacen.
 - Segundo clic: las células vecinas mueren.
 - Tercer clic: las células permanecen muertas.

Debido a que si hay dos o más células vecinas alrededor mueren.

2. Para crear una célula Conway ¿Cuáles son las adiciones necesarias en el diseño? ¿y los cambios? ¡Hágalos! Ahora codifique. Estas células van a ser azules. ¿Las pruebas son correctas?
 - En la clase AutomataCelular toco crear el método nace, cuando hay una celda vacía y tres células vivas.
 - Crear la clase Conway.
 - si, las pruebas son correctas.
3. Adicionen juntas en la fila cinco, una pareja de células Conway llámenlas john y horton.

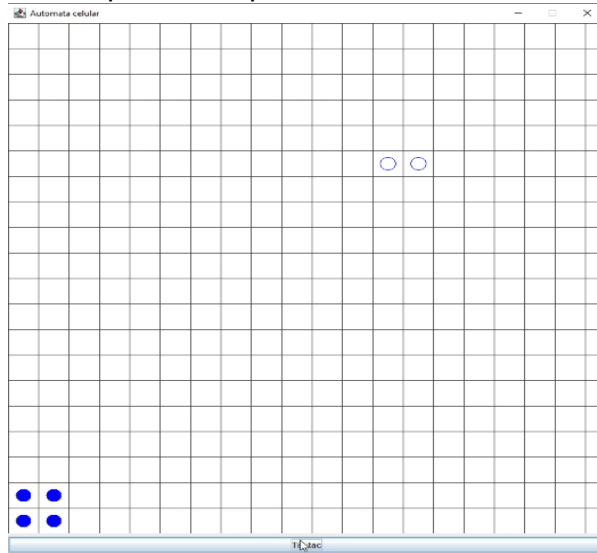


Ejecuten el programa, hagan tres clics en el botón **Tic-tac** y capturen la pantalla final.

¿Qué pasa? ¿es correcto?

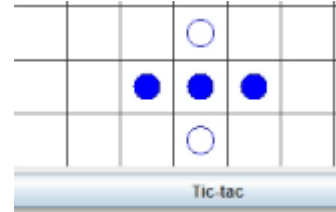
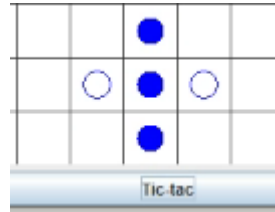
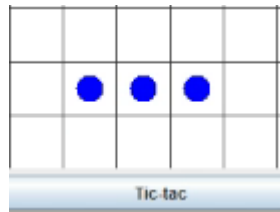
las células mueren, porque cada una tiene un vecino, mueren de soledad.

4. Adicionen en la esquina inferior izquierda un Bloque y ejecuten la aplicación, ¿qué pasa? ¿queda estático? Capture una pantalla. No olviden escribir la prueba correspondiente.



Las células siguen vivas, debido a que el bloque es un patrón estático.

5. Adicionen en la parte central inferior un Parpadeador (con espacio para parpadear) y ejecuten la aplicación, ¿qué pasa? ¿parpadea? Capture dos pantallas de parpadeo.



Al ejecutar el primer tictac nacen tres células, con el segundo tictac nacen dos células y en el tercer tictac también, donde se evidencia el parpadeo del patrón de los bloques.

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

1. Revise las opciones de BlueJ para empaquetar su programa entregable en un archivo .jar. Genere el archivo correspondiente.
2. Consulte el comando java para ejecutar un archivo jar. ejecutennlo ¿qué pasa?
java -jar "nombredelarchivo".jar
3. ¿Qué ventajas tiene esta forma de entregar los proyectos? Explique claramente
 - El paquete final es solo un archivo con el ejecutable de la app, genera seguridad para el programador ya que no se ve el código fuente, hace fácil el uso y es un archivo que ocupa menos espacio de memoria.

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

1. *Investiguen los comandos para moverse en la estructura de directorios: crear, borrar, listar su contenido y copiar o eliminar un archivo.*
 - Crear: MKDIR, MD.
 - Borrar: DEL
 - Listar: DIR
 - Copiar: COPY
 - Eliminar archivos: DEL, ERASE
 - Eliminar directorios: RD
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*

automata

src

aplicacion

presentacion

pruebas

```
E:\UNIVERSIDAD\7 semestre\POOB\lab3>mkdir automata

E:\UNIVERSIDAD\7 semestre\POOB\lab3>cd automata

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata>mkdir src

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata>mkdir src
Ya existe el subdirectorio o el archivo src.

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata>cd src

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata\src>mkdir aplicacion










E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata\src>mkdir presentacion

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata\src>mkdir pruebas

E:\UNIVERSIDAD\7 semestre\POOB\lab3\automata\src>
```

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

```
C:\Users\Daniel>xcopy "D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\*.java" D:\Descargas\Torrent/E
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\AutomataCelular.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\AutomataCelularTest.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Barrera.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Celula.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Conway.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Elemento.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Infeccioso.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Izquierdosa.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\aplicacion\Sociable.java
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\presentacion\AutomataGUI.java
10 archivo(s) copiado(s)
```

Nombre	Fecha de modificación	Tipo	Tamaño
 AutomataCelular.java	11/03/2020 11:20 p. m.	IntelliJ IDEA Com...	5 KB
 AutomataCelularTest.java	11/03/2020 9:21 p. m.	IntelliJ IDEA Com...	4 KB
 Barrera.java	11/03/2020 11:01 p. m.	IntelliJ IDEA Com...	2 KB
 Celula.java	11/03/2020 11:04 p. m.	IntelliJ IDEA Com...	3 KB
 Conway.java	11/03/2020 11:19 p. m.	IntelliJ IDEA Com...	2 KB
 Elemento.java	11/03/2020 10:58 p. m.	IntelliJ IDEA Com...	1 KB
 Infeccioso.java	11/03/2020 11:20 p. m.	IntelliJ IDEA Com...	2 KB
 Izquierdosa.java	11/03/2020 11:19 p. m.	IntelliJ IDEA Com...	1 KB
 Sociable.java	11/03/2020 11:20 p. m.	IntelliJ IDEA Com...	2 KB












Estructura de proyectos java [En lab03.doc]

En java los proyectos se estructuran considerando tres directorios básicos.

automata

src
bin
doc

1. *Investiguen los archivos que deben quedar en cada una de esas carpetas y la organización interna de cada una de ellas.*
 - src: Están los códigos fuentes *.java
 - bin: Están los bytecodes de los códigos fuentes *.class
 - docs: Tiene la documentación.
2. *¿Qué archivos debería copiar del proyecto original al directorio bin? ¿Por qué? Cópielos y consulte y capture el contenido del directorio que modificó.*
 - Los archivos .class, ya que son los elementos compilados de la clase

Nombre	Fecha de modificación	Tipo	Tamaño
 AutomataCelular.class	11/03/2020 9:40 p. m.	Archivo CLASS	3 KB
 AutomataCelularTest.class	11/03/2020 9:40 p. m.	Archivo CLASS	3 KB
 AutomataGUI.class	11/03/2020 9:56 p. m.	Archivo CLASS	3 KB
 Barrera.class	11/03/2020 9:40 p. m.	Archivo CLASS	2 KB
 Celula.class	11/03/2020 9:40 p. m.	Archivo CLASS	2 KB
 Conway.class	11/03/2020 9:40 p. m.	Archivo CLASS	2 KB
 Elemento.class	11/03/2020 6:52 p. m.	Archivo CLASS	1 KB
 FotoAutomata.class	11/03/2020 9:56 p. m.	Archivo CLASS	2 KB
 Infeccioso.class	11/03/2020 9:40 p. m.	Archivo CLASS	2 KB
 Izquierdosa.class	11/03/2020 9:40 p. m.	Archivo CLASS	1 KB
 Sociable.class	11/03/2020 9:40 p. m.	Archivo CLASS	2 KB

Comandos de java

1. *Consulte para qué sirven cada uno de los siguientes comandos:*
 - javac : Este comando se usa para compilar los *.java
 - java: Este comando se usa para ejecutar los *.java
 - javadoc: Este comando se usa para generar la documentación de los *.java
 - jar: Comprime proyectos java en un solo archivo y también ejecuta estos archivos.

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

```
C:\Users\Daniel>java
Usage: java [options] [mainclass] [args...]
       (to execute a class)
or java [options] -jar <jarfile> [args...]
       (to execute a jar file)
or java [options] -m <module>[/<mainclass>] [args...]
       (to execute a module)
or java [options] -module <module>[/<mainclass>] [args...]
       (to execute the main class in a module)
or java [options] <sourcefile> [args]
       (to execute a single source-file program)

Arguments following the main class, source file, -jar <jarfile>,
-m or -module <module>[/<mainclass>] are passed as the arguments to
main class.

where options include:
--cp <class search path of directories and zip/jar files>
--classpath <class search path of directories and zip/jar files>
--class-path <class search path of directories and zip/jar files>
        A ; separated list of directories, ZIP archives,
        and ZIP archives to search for class files.
-p <module path>
--module-path <module path>...
        A ; separated list of directories, each directory
        is a directory of modules.
--upgrade-module-path <module path>...
        A ; separated list of directories, each directory
        is a directory of modules that replace upgradeable
        modules in the runtime image.
--add-modules <module name>[,<module name>... ]
        Root modules to resolve in addition to the initial module.
        <module name> can also be ALL-DEFAULT, ALL-SYSTEM,
        ALL-MODULE-PATH.
--list-modules
        list observable modules and exit
-d <module name>
--describe-module <module name>
        describe a module and exit
--dry-run
        create VM and load main class but do not execute main method.
        The --dry-run option may be useful for validating the
        command-line options such as the module system configuration.
--validate-modules
        validate all modules and exit
        The --validate-modules option may be useful for finding
        conflicts and other errors with modules on the module path.
-Dname=value
        set a system property
--verbose[[:class|module|jni]]
        enable verbose output for the given subsystem
-version
        print product version to the error stream and exit
--version
        print product version to the output stream and exit
-showversion
        print product version to the error stream and continue
```

```
C:\Users\Daniel>javac
Usage: javac [options] [source files]
where possible options include:
-@filename
        Read options and filenames from file
-encoding <encoding>
        Specify character encoding used by source files
--module-path <module path>
        Specify where to find input source files for multiple modules
--module-source-path <module-source path>
        Specify where to find application modules
--processor-path <path>
        Specify where to find annotation processors
--processor <processor>
        Specify where to place generated class files
--release <release>
        Output source locations where deprecated APIs are used
--enable-preview
        Enable preview language features. To be used in conjunction with either -source or --release.
--encoding <encoding>
        Specify character encoding used by source files
--source <source>
        Specify where to place generated native header files
--help, -h, -?
        Print this help message
--help-extra, -X
        Print help on extra options
--implicit <name>
        Specify whether or not to generate class files for implicitly referenced files
--limit-modules <module>[,<module>... ]
        Limit the universe of observable modules
--module <module>[,<module>... ] -m <module>[,<module>... ]
        Compile only the specified module(s), check timestamps
--module-path <path>
        Specify where to find application modules
--module-source-path <module-source path>
        Specify where to find input source files for multiple modules
--module-version <version>
        Specify version of modules that are being compiled
--nowarn
        Generate no warnings
--parameters
        Generate metadata for reflection on method parameters
--proc <mode>[only]
        Control whether annotation processing and/or compilation is done.
        processor <class>[,<class>... ]
        Names of the annotation processors to run; bypasses default discovery process
--processor-module-path <path>
        Specify a module path where to find annotation processors
--processor-path <path>
        Specify where to find annotation processors
--profile <profile>
        Check that API used is available in the specified profile
```

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

```
C:\Users\Daniel>java -version
java version "13.0.2" 2020-01-14
Java(TM) SE Runtime Environment (build 13.0.2+8)
Java HotSpot(TM) 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)

C:\Users\Daniel>javac -version
javac 13.0.2
```

Compilando

1. Utilizando el comando javac, desde el directorio raíz (desde autómatas con una sola 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.

El comando es: 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?

En el directorio bin en la raíz del proyecto, aparecieron dos directorios, aplicación y presentación, los cuales contienen los archivos .class del proyecto.

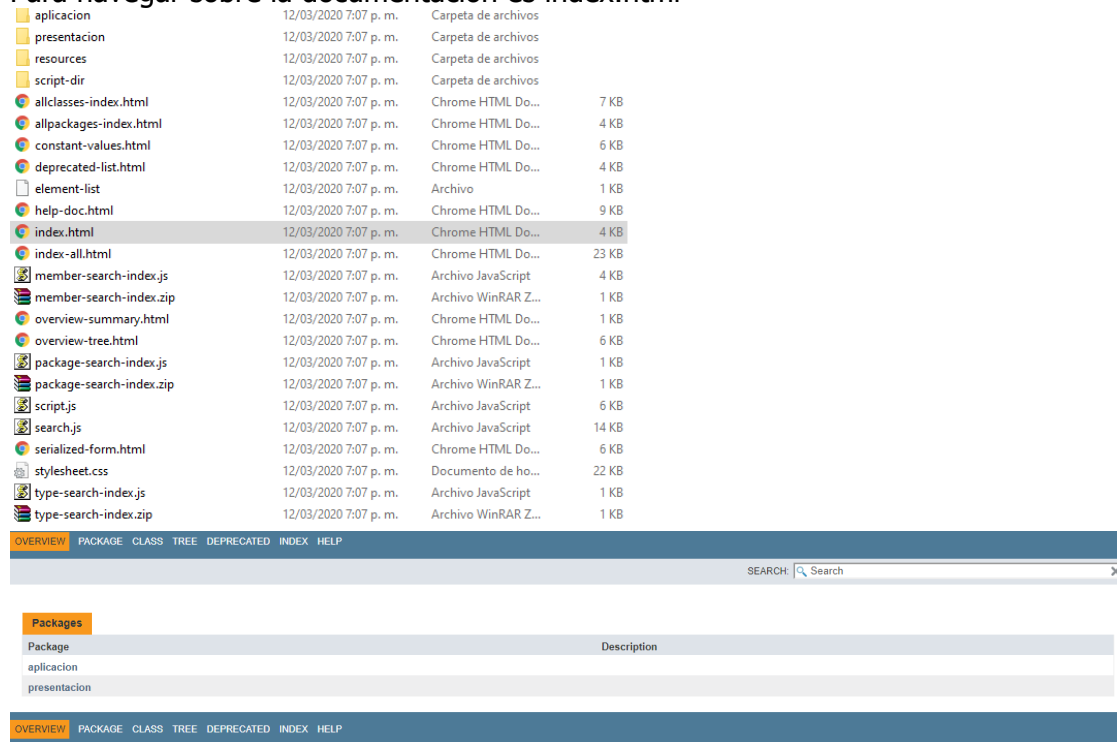
Documentando

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

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

Para navegar sobre la documentación es index.html



Ejecutando

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

Para ejecutar: java -cp bin presentacion.AutomataGUI

La sintaxis del comando es: java -cp /path/to/bin <packageName>.<MainClassName>

Probando [En lab03.doc]

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

El paquete tuvo que descargarse manualmente desde internet y se colocó en una carpeta llamada jars. Para incluirlo en la compilación se utiliza el comando –cp.

```
javac -d bin -cp jars\junit-4.13.jar src\aplicacion\*.java src\pruebas\*.java
src\presentacion\*.java
```

- 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>

```
javac -d bin -cp junit-4.13.jar;hamcrest-core-1.3.jar;bin org.junit.runner.JUnitCore
aplicacion.AutomataCelularTest
```

- Pegue en su documento el resultado de las pruebas

```
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata>java -cp jars\junit-4.13.jar;jars\hamcrest-core-1.3.jar;bin org.junit.runner.JUnitCore aplicacion.AutomataCelularTest
JUnit version 4.13
.....
Time: 0.034
OK (6 tests)
```

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 para empaquetar: jar cfe AutomataGUI.jar presentacion.AutomataGui aplicacion*.class presentacion*.class

```
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\bin>jar cfe AutomataGUI.jar presentacion.AutomataGui aplicacion\*.class presentacion\*.class
```

2. ¿Cómo se ejecuta el proyecto empaquetado?

```
D:\Documents\Daniel\Trabajos\13 - Trece\POOB\Laboratorio 3\automata\bin>java -jar AutomataGUI.jar
```

RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes?
(24/ 24)
2. ¿Cuál es el estado actual de laboratorio? ¿Por qué? (Para cada método incluya su estado)
El laboratorio se completó exitosamente.
3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importante?
Collective Ownership y integrate code at a time son las practicas del laboratorio y son importantes y que estas nos ayudan a organizarnos mejor como un equipo; a lograr un soporte mayor de cada uno; y el integrarnos para codificar juntos un mismo código más completo.
4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?
Finalizar el laboratorio en el tiempo estipulado, ya que fue un laboratorio muy extenso.
5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Trabajamos a pares, lo cual nos da a los dos un extenso conocimiento del proyecto, nos ayuda a conocer que es y cómo funciona.