PROGRAMACIÓN ORIENTADA A OBJETOS
Herencia e interfaces
ADEMAS Java desde consola
Septiembre 2015
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 : The project is divided into iterations.
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 afabéticamente.
- ➔ En el foro de entrega deben indicar el estado de avance de su laboratorio y los problemas pendientes por resolver.
- ➔ Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada en los espacios preparados para tal fin.

Contexto

Un autómata célular (A.C.) 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 caractísticas son:

1. La 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.

(ver wikipedia http://es.wikipedia.org/wiki/Automata_celular)

Conociendo. Arquitectura general. [En lab03.doc y automata.asta]

1. En el directorio descarguen los archivos contenidos en automata.zip. Revisen la aplicación ¿Qué nuevos componentes encuentran? ¿Qué creen que hacen?
2. Consulten el significado de las palabras package e import de java. ¿Qué es un paquete? ¿Para qué sirve? Explique su uso con un ejemplo del proyecto.
3. Inicie el diseño con un diagrama de paquetes en el que se presente estos componentes y las relaciones entre ellos.
4. Revise el contenido del directorio de trabajo y sus subdirectorios. Describan su contenido. ¿Qué coincidencia hay entre paquetes y directorios?

Conociendo. Detalles [En lab03.doc y automata.asta]

1. Revisen el código de la aplicación a) ¿Cuántas clases tiene en total? b) ¿Cuántas tienen fuentes? c) ¿Cuál es la clase ejecutiva?[1] ¿Por qué? d) ¿Cuál es la dimensión lógica de la rejilla del automata? ¿Por qué?
2. Ejecuten el programa. ¿Qué funcionalidades ofrece? ¿Qué hace actualmente? ¿Por qué?
3. Generen y revisen la documentación de la aplicación . a) ¿Qué componente falta? ¿por qué? ¿dónde si está? b) ¿Qué le podemos pedir al automata? c) ¿Cómo podemos saber si en una posición de un automata hay o no una célula? d) ¿Qué puede hacer una Celula? e) ¿Cómo podemos saber si una célula específica está viva?

1 Responsable de ejecutar la aplicación

f) ¿Cuáles acciones deben hacer igual y cuáles pueden variar en las células heredadas? ¿Por qué? f) ¿Cuáles atributos pueden y no pueden manipularce en todas las células heredadas? ¿Por qué?

4. Usando ingeniería reversa, presenten el diseño de la aplicación actual.
5. ¿Qué observaciones haría al respecto?

Ciclo 1. Dejando que el tiempo pase [En lab03.doc,automata.asta y *.java]

1. Si tenemos seguidas dos células normales vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriban la prueba correspondiente. ¿Es correcto?
2. En el método `main` creen dos células juntas en la fila 1, llámelas `daisy` y `minnie`. Ejecute el programa y capture una pantalla significativa. ¿Qué pasa ahora? Considerando el código de la Celula, ¿qué debería pasar?
3. En este punto vamos a construir el método que atiende el clic del botón ***Tic-Tac*** de la interfaz: el método llamado ticTac`()` de la clase `automata`, que usa dos métodos base `decidan()` y `cambien()`. Presente el diseño completo de este método y escriban el código de los métodos correspondientes. Ejecuten el programa y hagan tres clic en el botón ***Tic-tac***. ¿Qué hacen `daisy` y `minnie`.? Capture la pantalla inicial y la final.

Ciclo 2. Adicionando Izquierdosas [En lab03.doc,automata.asta y *.java]

El objetivo de este punto es incluir en la pista algunas células "izquierdosas". Estas células son de color rojo y deciden morir si no hay una célula viva a su izquierda (este).

1. Si tenemos aisladas y seguidas dos células izquierdosas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriban la prueba correspondiente.
2. Desarrollen la célula `Izquierdosa` (NO OLVIDE MDD y TDD) ¿Cuáles fueron las adiciones necesarias en el diseño? ¿y los cambios? ¿cuántas pruebas son correctas?
3. Para aceptar la célula `Izquierdosa` en automata, ¿debe cambiar en el código del `automata` en algo? ¿por qué?
4. Adicionen juntas una pareja de células izquierdosas en la fila 3, llámenlas `marx` y `hegel`, ejecuten el programa hagan dos clics en el botón **`Tic-tac`** y capturen la pantalla. ¿Qué pasa? ¿es correcto?

Ciclo 3. Adicionando Pobladoras [En lab03.doc,automata.asta y *.java]

El objetivo de este punto es incluir en la pista algunas células "pobladoras". Estas células son de color naranja, a los tres instantes están listas para poblar con dos nuevas células una izquierdosa al este y una propia al norte, si es posible. Después de poblar mueren.

1. Si tenemos asiladas y seguidas dos células pobladoras vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriban la prueba correspondiente.
2. Desarrollen la célula `Pobladora` (NO OLVIDE MDD y TDD) ¿Cuáles fueron las adiciones necesarias en el diseño? ¿y los cambios? ¿cuántas pruebas son correctas?
3. Para aceptar la célula Pobladora en automata, ¿debe cambiar en el código del `automata` en algo? ¿por qué?
4. Adicionen juntas una pareja de células pobladoras, en la fila 5, llámenlas `susanita` y `felipito`, ejecuten el programa con algunos clics en el botón **`Tic-tac`** y capturen una pantalla significativa. ¿Qué pasa? ¿es correcto?

Ciclo 4. Adicionando Necias [En lab03.doc,automata.asta y *.java]

El objetivo de este punto es permitir recibir en el automa células necias. Estas células son de color rosado y cuando les piden que decidan cambian y cuando le piden que cambien deciden.

1. Si tenemos seguidas dos células necias vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriban la prueba correspondiente.
2. Desarrollen la célula `Pobladora` (NO OLVIDE MDD y TDD) ¿Cuáles fueron las adiciones necesarias en el diseño? ¿y los cambios? ¿cuántas pruebas son correctas?
3. Para aceptar la célula `Necia` en automata, ¿debe cambiar en el código del `automata` en algo? ¿por qué?
4. Ahora si adicionen juntas en la fila cinco, una pareja de células necias llámenlas `homero` y `bard`. Ejecuten el programa, hagan cinco clics en el botón **`Tic-tac`** y capturen la pantalla final.¿Qué pasa? ¿es correcto?

Caso 5. Diseñando y creando una nueva célula [En lab03.doc,automata.asta y *.java]

Propongan, dEscriban e implementen una nueva célula Incluyan una pareja de ellas en la fila siete con el nombre de ustedes. (NO OLVIDE MDD y TDD)

Caso 6. El Juego de la vida [En lab03.doc,automata.asta y *.java]

El juego de la vida es el mejor ejemplo de un autómata célular, diseñado por el matemático británico John Horton Conway en 1970. Un automata célular 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 evoluciona a lo largo de unidades de tiempo y los cambios dependen del número de células vecinas vivas:

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

1. Si tenemos seguidas dos células Conway vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriban la prueba correspondiente. [En
2. Desarrollen la célula `Pobladora` (NO OLVIDE MDD y TDD) ¿Cuáles fueron las adiciones necesarias en el diseño? ¿y los cambios? ¿cuántas 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?
4. Adicionen en la esquina inferior izquierda un Bloque y ejecuten la aplicación, ¿qué pasa? ¿queda estático? Capture una pantalla.
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.
6. Adicionen en la esquina inferior derecha un Planeador y ejecuten la aplicación, ¿qué pasa? ¿viaja? Capturen la pantalla inicial y una que muestre su avance.

Empaquetando la versión final para el usuario.

1. Empaque la versión final para usuario. [En automata.jar]
2. Realice sus pruebas de aceptación final y documéntelas [en lab03.doc]

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.

2. Investiguen los comandos para moverse en la estructura de directorios: crear, borrar, listar su contenido y copiar o eliminar un archivo.
3. 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
```

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

Estructura de proyectos java [En lab03.doc]
En java los proyectos se estructuran considerando tres directorios básicos.

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

Comandos de java [En lab03.doc]

1. Consulte para qué sirven cada uno de los siguientes comandos:
   `javac`
   `java`
   `javadoc`
   `jar`
2. Cree una sesión de consola y consulte en línea las opciones de los comandos java y javac. Capture las pantallas.
3. Busque la opción que sirve para conocer la versión a que corresponden estos dos comandos. Documente el resultado.

Compilando [En lab03.doc]

1. Utilizando el comando `javac`, **desde el directorio raiz (desde `automata` 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.
2. Revise de nuevo el contenido del directorio de trabajo y sus subdirectorios. ¿Cuáles nuevos archivos aparecen ahora y dónde se ubican?

Documentando [En lab03.doc]

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

Ejecutando [En lab03.doc]

5. Empleando el comando `java`, desde el directorio raiz, ejecute el programa. ¿Cómo utilizó este comando?

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. ¿Cómo se incluye un paquete para compilar? ¿Qué instrucción completa tuvo que dar a la consola para compilar?
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
3. Pegue en su documento el resultado de las pruebas

Empaquetando [En lab03.doc]

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

RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)
2. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importante?
3. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?
4. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?