PROGRAMACIÓN ORIENTADA A OBJETOS
Herencia e interfaces
ADEMAS Java desde consola
2020-1
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 alfabéticamente.
- ➔ 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.

DESARROLLO

Contexto

Un autómata célular (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 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 [En lab03.doc y automata.asta]

1. En el directorio descarguen los archivos contenidos en automata.zip. Revisen el código de la aplicación a) ¿Cuántos paquetes tiene? b) ¿Cuántas clases tiene en total? ¿Cuántas tienen fuentes? c) ¿Cuál es la clase ejecutiva? ¿Por qué?
2. Ejecuten el programa. ¿Qué funcionalidades ofrece? ¿Qué hace actualmente? ¿Por qué?

Arquitectura general. [En lab03.doc y automataasta]

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.
2. Revise el contenido del directorio de trabajo y sus subdirectorios. Describa su contenido. ¿Qué coincidencia hay entre paquetes y directorios?
3. Inicie el diseño con un diagrama de paquetes en el que se presente los componentes y las relaciones entre ellos.
   En astah, crear un diagrama de clases (cambiar el nombre por Package Diagram0)

---

1 http://es.wikipedia.org/wiki/automataCelular_célular

Arquitectura detallada. [En lab03.doc y automataasta]

1. Usando ingeniería reversa prepararen 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? ¿Asociado a qué clase? ¿Por qué?

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

(NO OLVIDE BDD - MDD)

3. Estudie la clase AutomataCelular ¿Qué tipo de colección usa para albergar los elementos? ¿Puede recibir células? ¿Por qué?
4. Estudie el código de la clase Celula, ¿qué otras clases la definen? ¿cómo?
5. Todas las células ¿qué saben hacer? ¿qué no puede hacer distinto? ¿qué debe aprender a hacer? Justifique su respuesta.
6. Por comportarse como un elemento, ¿qué sabe hacer? ¿qué no puede hacer distinto? ¿qué debe aprender a hacer? Justifique su respuesta.
7. Considerando lo anterior, una Celula ¿de qué color es? ¿cómo decide? ¿cómo cambia? Justifiquen sus respuestas.
8. 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 todos las células? Capturen una pantalla significativa.
9. 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`**? Escriba la prueba correspondiente.
10. Construyan el método. ¿Es correcto?
11. Ejecuten el programa y hagan tres clic en el botón. ¿Como quedan las células? Capturen una pantalla significativa.

Ciclo 2. Incluyendo a las células izquierdosas [En lab03.doc y automataasta]

(NO OLVIDE BDD - MDD)

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

4. Si tenemos seguidas dos células izquierdosas vivas en la misma fila, ¿qué debería pasar en el primer, segundo y tercer clic? ¿por qué? Escriba la prueba correspondiente.
5. ¿Cuáles son las adiciones necesarias en el diseño? ¿y los cambios? ¡Hágalos! ¿cuáles métodos se sobre-escriben *(overriding)*? Ahora escriba el código correspondiente a la célula `Izquierdosa` ¿Las pruebas son correctas?
6. Para aceptar la célula `Izquierdosa` en AutomataCelular, ¿debe cambiar en el código del `AutomataCelular` en algo? ¿por qué?
7. 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.
8. Construyan el método. ¿Es correcto?
9. Ejecuten el programa y hagan dos clic en el botón. ¿Como quedan las células? Capturen una pantalla significativa.

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

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

(NO OLVIDE BDD – MDD)

1. Construyan la clase `Barrera` para poder adicionaría en el `AutomataCelular` ¿qué hicieron?
2. Para aceptar este elemento , ¿debe cambiar en el código del `AutomataCelular` en algo? ¿por qué?
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.
4. Construyan el método. ¿Es correcto?
5. Ejecuten el programa y hagan tres clics en el botón. Capturen una pantalla significativa. ¿Qué pasa? ¿es correcto?

Ciclo 4. Nueva Celula: 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 un nuevo tipo de células.
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.

Ciclo 5. Nuevo elemento: Proponiendo y diseñando

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

(NO OLVIDE BDD - MDD)

- Propongan, describan e Implementen un nuevo tipo de elemento
- Incluyan un par de ellos con el nombres semánticos. ejecuten el programa con dos casos significativos. Explique la intención de cada caso y Capturen las pantallas correspondientes.

Caso 6. El Juego de la vida

El juego de la vida es el mejor ejemplo de un autómata célular, diseñado por el matemático británico John Horton Conway en 1970. Un AutomataCelular 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 AutomataCelular 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 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 AutomataCelular.

3. 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.
4. 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?
5. 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?
6. 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.
7. Adicionen en la parte central inferior un Parpadeador (con espacio para parpadear) y ejecuten la aplicación, ¿qué pasa? ¿parpadea? Capture dos pantallas de parpadeo. No olviden escribir la prueba correspondiente.

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

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.
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
```

3. 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]

6. 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. ¿Cuál es el estado actual de laboratorio? ¿Por qué? (Para cada método incluya su estado)
3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importante?
4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?
5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?