Tarea 1 Enunciado – Sokoban

Descripción del problema

Una empresa independiente de desarrollo de videojuegos casuales necesita implementar el código para manejar la lógica de su próximo videojuego. Debido a que la empresa actualmente solo cuenta con un diseñador de juegos y un equipo de ilustradores y animadores, se requiere contratar un equipo de desarrolladores para que implementen el código que se encarga de la lógica del juego. Este equipo de desarrollo es su equipo.

El videojuego que la empresa desea desarrollar es una nueva versión del juego clásico de rompecabezas (puzzle) *Sokoban*. *Sokoban* fue inventado en 1981 por Hiroyuki Imabayashi.

La dinámica del juego consiste en que un encargado de un almacén debe ordenar cajas colocándolas en los puntos designados. El almacén es representado por un tablero compuesto de cuadrados. Los cuadros pueden representar piso o paredes. Sobre el piso se encuentran las cajas y las señales que indican la posición donde la caja debe colocarse.

Para ordenar las cajas, el jugador puede moverse horizontalmente y verticalmente sobre posiciones de piso vacías o posiciones designadas (que se encuentren vacías). Además, el jugador puede empujar las cajas siempre y cuando no haya un obstáculo (una pared u otra caja) que impida mover la caja hacia esa dirección. El jugador no puede jalar una caja, únicamente empujarla.

El siguiente enlace muestra un ejemplo de *Sokoban*: https://en.wikipedia.org/wiki/Sokoban.

La empresa de videojuegos requiere que su equipo de desarrolladores escriba un programa que verifique el estado del juego cada vez que el jugador realiza un movimiento. Los aspectos que se deben verificar son:

- Si el jugador ganó el juego.
- Las posiciones actuales de las cajas.
- Las posiciones de las cajas bloqueadas.
- Los movimientos válidos que el jugador puede realizar el próximo turno.

Para mantener el programa eficiente, el tablero de Sokoban se codifica de la siguiente manera:

Tabla 01. Símbolos para representar un tablero.

Símbolo	**Significado**
#	Pared.
X	Una posición donde se debe colocar una caja, con una caja.
*	Caja sobre cualquier otra posición que no sea la final.
O	Una posición donde se debe colocar una caja, pero sin la caja.
@	El encargado del almacén.
.	Piso vacío.

Su programa debe leer de la entrada estándar el número de filas, número de columnas, número de cajas, y un tablero del estado actual del juego.

Tabla 02. Ejemplo de entrada:

```
5 5 2
```

```
#####
#*.O#
#X.O#
#..@#
#####
```

Luego, debe realizar las verificaciones que se le piden y generar un reporte del estado del juego. El reporte debe de imprimirse en la salida estándar y debe tener el siguiente formato.

Tabla 03. Ejemplo de salida:

```
           Victoria: No
              Cajas: r01c01 r02c01*
   Cajas Bloqueadas: r01c01
Movimientos válidos: N:r02c03 E:- S:- O:r03c02
```

La **condición de victoria** ocurre cuando todas las cajas están ordenadas. En otras palabras, cuando todas las cajas están sobre los puntos designados. Si la condición se cumple su programa de reportar que si se cumple la condición de victoria.

Las **posiciones actuales de las cajas** deben reportar las coordenadas de todas las cajas siguiendo el formato que se muestra en la Tabla 03. La "r" indica la fila (row) y la "c" la columna en que se encuentra una caja. Además, se debe resaltar con un "*" al final de la coordenada las cajas que se encuentran sobre un punto designado.

Las **posiciones actuales de las cajas bloqueadas** corresponden a las coordenadas de las cajas que no pueden moverse más y que no se encuentran en una posición designada. El reporte de las cajas bloqueadas debe seguir el mismo formato mencionado en el punto anterior.

Los **movimientos válidos** reportan las casillas en las que se puede mover el jugador el siguiente turno. Se deben reportar las cuatro casillas (arriba, abajo, izquierda y derecha) utilizando los puntos cardinales (norte, este, sur, oeste) en el orden de las manecillas del reloj. Si la posición es válida, se reporta la coordenada, de lo contrario, se reporta el signo "-".

Finalmente, si el programa recibe valores faltantes, menores o igual a cero, caracteres o símbolos en la primera línea, el programa debe imprimir "entrada invalida". Si el programa encuentra en la matriz que representa el tablero, valores faltantes, símbolos u otros caracteres que no son los que se mencionan en la Tabla 1, el programa debe imprimir "matriz invalida".

Control de versiones

El proyecto será realizado en parejas en un repositorio de control de versiones. Ambos participantes deben trabajar todas las fases del proceso (análisis, diseño, programación, documentación, pruebas) y deben dejar evidencia en el historial de *commits*. La nota de un participante será proporcional a la cantidad y calidad de los *commits* que realizó.

Haga buen uso del repositorio de control de versiones. Cada commit debe tener un cambio y un mensaje significativo, y "no romper el build". No haga "megacommits" que implementan muchas funcionalidades distintas y acumuladas en varios días de desarrollo.

Por ninguna razón deben agregar archivos binarios a control de cambios o que hayan sido generados a través de un proceso automatizado, por ejemplo, los archivos de la carpeta build, ejecutables, bibliotecas, documentación generada por Javadoc. En su lugar, deben tener un archivo .gitignore adecuado. Deben agregar al profesor del curso como colaborador a su repositorio de control de versiones.

Las carpetas y archivos que se deben agregar al control de versiones son los siguientes:

Tabla 04. Carpetas y archivos el proyecto.

Recurso	Descripción	Versionado
.gradle/	Archivos de Gradle generados automáticamente.	NO
.idea/	Archivos de IntelliJ generados automáticamente.	NO
design/	Contiene el pseudocódigo de sus procedimientos.	SI
doc/	Contiene su documentación generada por Javadoc.	NO
gradle/	Carpeta con archivos Gradle necesarios para construir el proyecto con la versión de Gradle con la que fue creado.	SI
src/	Archivos con su código (archivos fuente).	SI
tests/	Archivos con los casos de prueba facilitados por el profesor.	NO
build.gradle	Contiene configuraciones para construir el proyecto.	SI
gradlew	Contiene configuraciones para construir el proyecto.	SI
gradle.bat	Contiene configuraciones para construir el proyecto.	SI
settings.graddle	Contiene configuraciones para construir el proyecto.	SI
.gitignore	Indica cuales archivos y carpetas no deben ser incluidos al control de versiones.	SI

Análisis

En un archivo README.md en notación Markdown plasme el resultado de la fase de análisis. Si no conoce el formato Markdown, asegúrese de seguir un manual o buscar un tutorial, ya que la buena calidad del código también será evaluada. Por ejemplo, un párrafo en estas notaciones se forma al separar por dos cambios de línea y no por uno. Su documento de análisis debe incluir:

1. Una descripción del problema a resolver. Puede adaptar partes de este enunciado, pero no copiarlas exactamente igual, ya que el enunciado va dirigido al estudiantado, mientras que el README va dirigido a cualquier persona que visite el proyecto y quiera entender el problema que la solución resuelve.
2. Una sección de créditos, donde indica su nombre e información de contacto (correo institucional). Si utiliza recursos de terceros, como código fuente (por ejemplo, si utilizó código de StackOverflow), dé el respectivo crédito.

Diseño

Su proyecto debe traer una carpeta llamada design donde debe incluir el pseudocódigo de su solución. Durante la etapa de diseño debe enforcarse en la solución general, no es necesario enforcarse en los detalles del lenguaje Java. Utilice las 8 instrucciones básicas del paradigma procedimental (ver en Mediación Virtual) más la instrucción RETURN.

La solución de su pseudocódigo debe ser modular. Las verificaciones y la impresión del informe se deben realizar en procedimientos separados. Está practica le ayudará a dividir los problemas en problemas más pequeños y fáciles de manejar. Además, le ayudará a reutilizar código.

Documentación

Su código debe trae documentación interna siguiendo la convención de Javadoc vista en clase. La clase, atributos, procedimientos públicos y privados deben venir documentados. El siguiente ejemplo corto muestra cómo se espera la documentación de un procedimiento:

Tabla 05. Ejemplo de documentación Javadoc.

```
/**
 * El método {@code calcularMediana} calcula la mediana de un arreglo de
 * números reales.
 *
 * @param numElementos Es un número de tipo {@code int} que indica el tamaño
 *                                                  del arreglo.
 * @param arrayElementos Arreglo de {@code doubles} al que hay que sacar la
 *                                                   mediana.
 * @return Se retorna un {@code double} con el resultado de la mediana
 *  estadística de arreglo.
 */
public double calcularMediana(final int numElementos,
                              final double[] arrayElementos) { … }
```

Además, debe incluir documentación dentro del cuerpo de los procedimientos para explicar que hacen las instrucciones más significativas de su procedimiento.

Implementación y Pruebas

Su código debe de apegarse al estándar de Google y SUN versión modificada que estará disponible en Mediación Virtual. Además, debe utilizar nombres significativos para los procedimientos y las variables.

Asegúrese de verificar el funcionamiento de su programa con los casos de prueba (caja negra) que se adjuntan. Tiene que generar al menos una prueba de tamaño y dificultad significativa.

Evaluación

1. [10%] Buen uso de los commits, estructura de archivos y directorios, ignorar archivos autogenerados.
2. [10%] Análisis (README.md).
3. [10%] Diseño de la solución (pseudocódigo).
4. [10%] Documentación interna: Javadoc y documentación dentro de los procedimientos.
5. [50%] Correcta implementación del programa, nombres significativos, y pasar el CheckStyle (Sun y Google).
6. [10%] Pasar todos los casos de prueba (caja negra).

Cualquier acto de plagio será sancionado severamente.