Dpto: LSIIS - *Asignatura: Programación II*

Práctica Opcional: MiniAjedrez

Objetivo: El objetivo de esta práctica es la familiarización del alumno con la programación orientada a objetos en Java, concretamente la implementación de clases que heredan de una clase y el uso de estructuras de tipo pila.

Desarrollo del trabajo: La práctica se podrá realizar en grupos de dos alumnos o de forma individual.

Grupos: Los grupos estarán formados por dos alumnos matriculados en el mismo semestre. **El grupo será creado en el momento en el que se realice la primera entrega en la que el código suministrado por el alumno compile**. Será creado a partir de los datos incluidos en la notación (véase Consideraciones de Entrega). Una vez creado el grupo será **el mismo para el presente ejercicio y no habrá posibilidad alguna de cambio** por lo que se aconseja revisar bien el código para asegurarse de que los datos de los alumnos están bien puestos.

Desarrollo del código: La práctica entregada debe compilar en la versión 1.8 del J2SE de Oracle/Sun y debe ser compatible con **JUnit 4.12**. Los ficheros fuente a desarrollar es el que corresponde a las clases **GestionHistorial.java, PiezaAjedrez.java, Alfil.java, Peon.java, Rey.java, Reina.java, Caballo.java.**

Código auxiliar: La realización de esta práctica requiere la utilización de los ficheros auxiliares que se pueden encontrar dentro del fichero **MiniAjedrez.zip** que acompaña a este enunciado.

Autoevaluación: El alumno debe comprobar que su ejercicio no contiene ninguno de los errores explicados en el apartado Errores graves a evitar de este enunciado. **Si el ejercicio contiene alguno de estos errores, la calificación final de la práctica será penalizada en consecuencia**. Además, se debe comprobar que las pruebas JUnit suministradas (ver Apartado Pruebas) se ejecutan con éxito.

Entrega: La práctica se entregará a través de la página web:

http://triqui2.fi.upm.es/entrega/

Cuando la práctica se hace en grupo, el alumno del grupo que realice la primera entrega de una práctica será el que tenga que hacer todas las demás entregas de esa misma práctica.

Plazos: **El periodo de entrega finaliza el 25 de mayo a las 9:00 AM**. En el momento de realizar la entrega, la práctica será sometida a una serie de pruebas que deberá superar para que la entrega sea admitida. El alumno dispondrá de **un número máximo de 20 entregas**. El hecho de que la práctica sea admitida por el sistema no implicará que la práctica esté aprobada.

Material a entregar: El alumno tendrá que entregar los ficheros **GestionHistorial.java, PiezaAjedrez.java, Alfil.java, Caballo.java, Peon.java, Reina.java, Rey.java**. Estos ficheros deben compilar y funcionar con el código suministrado con esta práctica sin modificación alguna.

Evaluación: El peso de esta práctica en la nota final de la asignatura es el que indica la Guía de la Asignatura. **En la convocatoria extraordinaria no existirá la posibilidad de entregar esta práctica opcional**.

Detección automática de copias: Cada práctica entregada se comparará con el resto de las prácticas entregadas en todos los grupos de la asignatura. Esto se realizará utilizando un sofisticado programa de detección de copias.

Consecuencias de haber copiado: Se aplicarán las normas publicadas en el Aula Virtual (plataforma Moodle) de la asignatura.

1. Enunciado de la práctica: MiniAjedrez

En esta práctica se pide completar la implementación de algunas de las partes que componen un juego de ajedrez simplificado. Dicho juego de Ajedrez incluye buena parte de los movimientos habituales de las piezas del ajedrez, pero no incorpora algunos de los movimientos como el enroque del rey, la captura al paso del peón o la coronación del peón. Asimismo, la partida no termina con la existencia de un *jaque mate* (ni esto se comprueba), sino que requiere que el Rey sea efectivamente comido por el contrincante para que la partida termine. En la Figura 1 se puede observar la interfaz del juego.

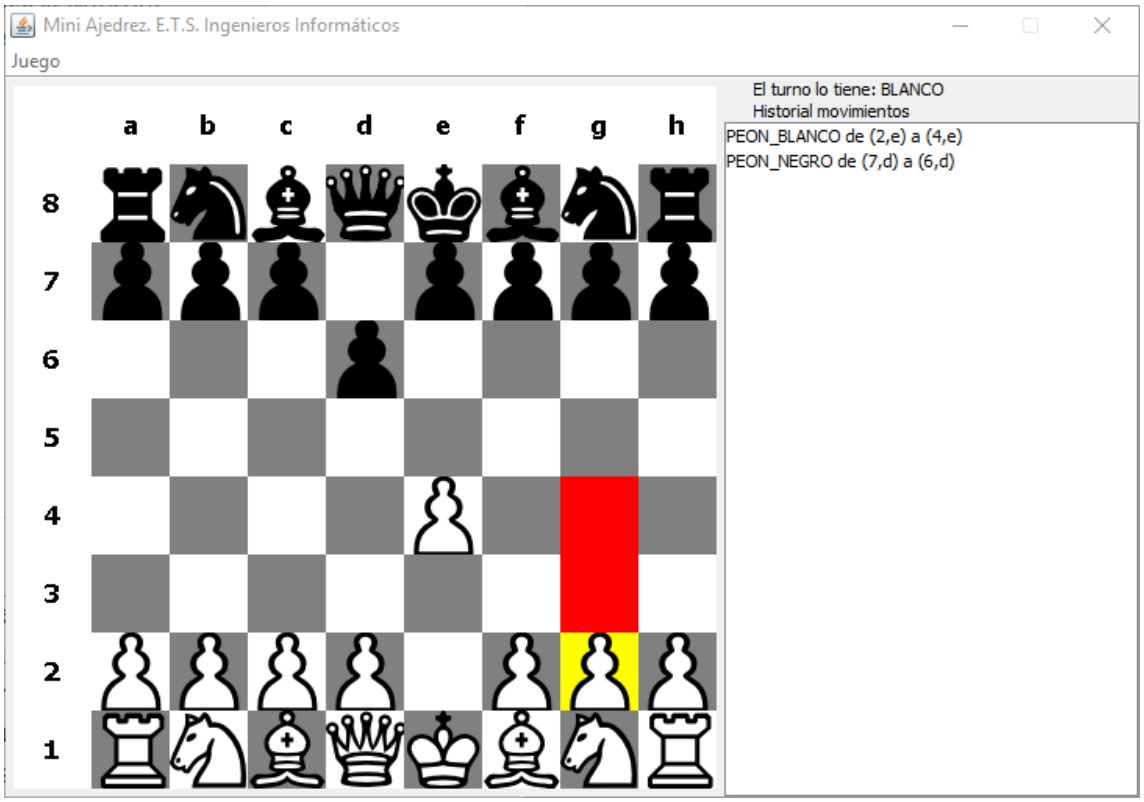

Figura 1 Captura de pantalla del Juego

Para mover una pieza el jugado pinchará sobre la pieza que desea mover, seleccionándose ésta en amarillo (como se ve en la Figura 1) y en rojo aparecerán casillas a las que puede mover la ficha según el estado de la partida (se recuerda que esta versión simplificada no comprueba si el rey está en jaque o va a estarlo tras un movimiento). Para mover la pieza, el usuario deberá pinchar de nuevo sobre alguna de las casillas en rojo. Las casillas que se presentan en rojo tienen en cuenta el estado del tablero, es decir, se muestran todos los movimientos que podría hacer la pieza desde la posición en la que está considerando el estado del tablero (sin tener presente si el rey está o no amenazado). La pieza debe calcular si podría o no transitar a una casilla, para eso la clase PiezaAjedrez proporciona un método de clase protegido llamado queHay, que permite determinar que hay en una casilla dada.

Las posiciones de las piezas se identifican de acuerdo con las posiciones definidas en el ajedrez (véase Figura 1) de forma que la casilla superior derecha es la (8, a) y la inferior

derecha la (1, h). En el historial de movimientos de la Figura 1 se ve como el peón blanco se ha movido de la posición (2, e) a la posición (4, e). Tal y como se ve en la Figura 1, las piezas negras empiezan siempre en la parte "alta" del tablero.

Para la implementación de este ejercicio se suministra el archivo **MiniAjedrez.jar** que contiene la implementación de una parte del juego. A continuación, daremos una breve descripción de algunas de las clases que os resultarán útiles para el desarrollo de la práctica:

En el juego participan un conjunto de objetos que representan las piezas del ajedrez:

- `Pieza`[1]: Clase abstracta de la que derivan todas las clases que representan las piezas de cualquier juego de tablero. El alumno debe leer la documentación de la clase para saber qué métodos debe utilizar y sobrescribir para conseguir el correcto funcionamiento del juego. La clase `Pieza` dispone de los métodos `getFila()` y `getColumna()` para consultar la posición (fila/columna) que ocupa la pieza y que debéis utilizar para implementar las clases que se detallan a continuación.
- `PiezaAjedrez`: Especializa la pieza con características de las piezas de una partida de Ajedrez. Su labor fundamental es comprobar que al mover las piezas no se salen de un tablero de ajedrez.
- `Torre`: Implementación de la Torre en el ajedrez, que se entrega ya implementada como inspiración para implementar el resto de las piezas.
- `Alfil`, `Caballo`, `Rey`, `Reina`, `Peón`: Clases que representan cada una de las piezas del ajedrez y extienden la clase `PiezaAjedrez`.

El diagrama de clases que representa estas clases se encuentra en la Figura 2.

Asimismo, el juego dispone de la posibilidad de deshacer y rehacer los movimientos realizados en la partida. De dicha gestión se encarga la clase:

- `GestorHistorial`: Clase que gestiona los movimientos realizados y permite deshacer y rehacer movimientos.

Como funcionalidad extra para facilitar las tareas depuración, el juego dispone de la opción de guardar y cargar las partidas. Dicha opción se encuentra disponible en el menú de la aplicación.

[1]El .class se encuentra en el jar suministrado con el enunciado

1.1. Trabajo del Alumno

En la Figura 2 se muestra el diagrama de clases de la jerarquía de piezas. De la clase Pieza (en naranja en el diagrama) se suministra el fichero .class (no se proporciona el código fuente) dentro de MiniAjedrez.jar. De la clase Torre (en amarillo en el diagrama) se entrega el código fuente a modo de ejemplo, pero este código no funcionará si el alumno no completa correctamente la clase PiezaAjedrez (en verde en el diagrama). Las clases Alfil, Caballo, Peón, Reina y Rey (clases en color azul en el diagrama) **deben ser creadas desde cero** en el paquete `progii.juegotablero.model.ajedrez.piezas` (el mismo paquete en el que se encuentra la clase Torre) y deberá implementarse el método `movimientosValidos`.

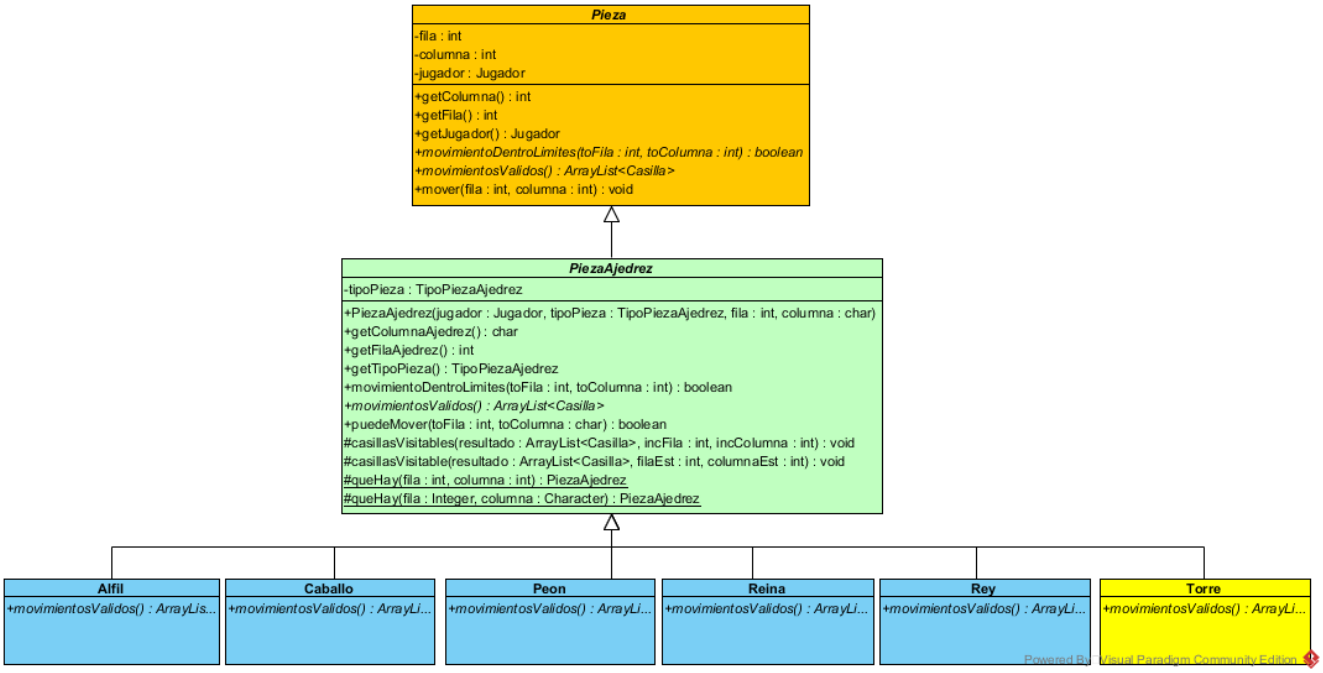

Figura 2 Diagrama de clases (Piezas)

- Clase **`PiezaAjedrez`**, que hereda de la clase Pieza y que debe estar en el paquete `progii.juegotablero.model.ajedrez`. El alumno debe implementar los siguientes métodos:
  - Sobrescribir el método `movimientoDentroLimites`. Este método debe comprobar que el movimiento se realiza dentro del tablero y que la posición a la que se quiere mover la pieza es diferente de la posición de origen de la pieza. Las posiciones se reciben siguiendo el criterio de una matriz. Recibe dos enteros donde la fila **0** y columna **0** representa la posición superior izquierda
  - Implementar el método `puedeMover`. Este método es similar al anterior, pero en este caso este método recibe la fila y la columna según se especifican en el

juego de ajedrez. En este caso la posición superior izquierda corresponde a la fila **8** y columna **a**.

- Implementar el método `casillasVisitables`. Este método protegido proporciona servicio al resto de piezas de forma que estas puedan calcular sus movimientos. Va a recorrer el tablero desde la posición inicial de la pieza hasta llegar a un extremo del tablero o encontrar una pieza. La forma en que se hace el recorrido del tablero viene determinada por incFila e incColumna. Insertará en resultado todas las casillas visitables, es decir, todas las casillas consecutivas (aplicando en cada iteración incFila e incColumna) que estén vacías, hasta llegar a un extremo del tablero o encontrar una pieza. En este último caso, si la pieza que ocupa la casilla es de distinto color también formará parte del resultado. En el método `movimientosValidos` de la clase **Torre** se puede observar cómo se utiliza el método `casillasVisitables` para obtener todas las casillas a las que se podría mover una torre desde su posición actual.

- Las clases **`Alfil`**, **`Caballo`**, **`Rey`**, **`Reina`**, **`Peon`**, del paquete `progii.juegotablero.model.ajedrez.piezas` en las que se precisa:

  - Sobrescribir el método `movimientosValidos`. Este método debe retornar una list.ArrayList<Casilla> con todas las casillas visitables por la pieza según el estado del tablero sin tener en cuenta la existencia de jaque, o la posibilidad de enroque o captura al paso de un peón. En el caso de que la pieza no pueda ser movida, el ArrayList retornado tendrá cero elementos. Este método nunca retorna **null**. Los movimientos válidos de cada una de las piezas los podéis encontrar en el siguiente enlace:

https://docs.kde.org/trunk5/es/extragear-games/knights/piece-movement.html

NOTA: Como el movimiento de una pieza depende del color del mismo (cuando se come o captura una pieza esta debe ser del color contrario), dicho color se puede consultar mediante el jugador al que pertenece la pieza con la siguiente comprobación:

```
this.getJugador().getId() == ControlJugadoresAjedrez.NEGRO
```

- La clase **`GestorHistorial`** del paquete `progii.juegotablero.model`. Dicha clase implementa la gestión del historial de movimientos para poder deshacer y rehacer los movimientos de las piezas. Dicha clase contendrá dos pilas, una en la que almacenarán los movimientos para que se puedan deshacer y otra que se encarga de almacenar los movimientos a rehacer. En la clase GestorHistorial se precisa implementar:

  1. Contendrá dos atributos: `pilaDeshacer` y `pilaRehacer`, que son dos pilas de tipo `stacks.Stack` que almacenarán los objetos de tipo `Movimiento` que se vayan produciendo en el desarrollo de la partida.

2. El constructor sin parámetros que inicializa las pilas `pilaDeshacer` y `pilaRehacer`.

3. El método `guardarMovimiento`, que recibe como parámetro de tipo `Movimiento` y lo almacena en la pila de deshacer. Debe comprobar si la pilaRehacer está vacía y en caso de que no sea así, debe eliminar todos los elementos de la pilaRehacer.

4. El método `deshacer`, que no recibe ningún parámetro, y deshace el último movimiento guardado, devolviendo el objeto de tipo `Movimiento` que se acaba de deshacer. Dicho movimiento debe ser quitado de la pila de deshacer y guardado en la pila de rehacer. En caso de que no haya movimientos que deshacer, se lanzará la excepción `MovimientoException` con el mensaje "No se puede deshacer porque no hay movimientos para deshacer".

5. El método `rehacer`, que no recibe ningún parámetro, y rehace el último movimiento que se hubiera deshecho, devolviendo el objeto de tipo `Movimiento` que se acaba de rehacer. Dicho movimiento debe ser quitado de la pila de rehacer y guardado en la pila de deshacer. En caso de que no haya movimientos que rehacer lanzará la excepción `MovimientoException` con el mensaje "No se puede rehacer porque no hay movimientos para rehacer".

2. Diseño de la aplicación

2.1. Especificación de los métodos

Las descripciones de las clases y los métodos están disponibles en los documentos html disponibles en la carpeta doc de los materiales para el alumno. El alumno debe leerse esta documentación y entender lo que hacen los métodos de las clases que necesite para la implementación de la práctica, así como de las clases que debe implementar.

3. Código de apoyo

El fichero **MiniAjedrez.zip** contiene todos los ficheros necesarios para el desarrollo de la práctica. Consta de los siguientes directorios y ficheros:

- Directorio **lib**: Contiene el jar con la parte que se da implementada del ajedrez.
- Directorio **doc**: documentación de las constantes públicas y los métodos públicos y protegidos de las clases. El fichero que debéis abrir es index.html.

- Directorio **src**: contiene los ficheros .java que se suministran ubicados en sus correspondientes paquetes (subdirectorios). Estos ficheros no deben ser cambiados de ubicación. Los archivos suministrados son:
  - progii/juegotablero/gui/MainAjedrez.java: Fichero que contiene el main y que ejecuta el juego y muestra la interfaz del juego
  - progii/juegotablero/model/ajedrez/PiezaAjedrez.java: plantilla para la implementación incompleta de la clase PiezaAjedrez.
  - progii/juegotablero/model/ajedrez/piezas/Torre.java: Clase que implementa la Torre. Si los métodos de PiezaAjedrez se implementan correctamente, el JUnit para la torre pasará todas las pruebas
  - progii/tests/TestTorre.java: JUnit para la torre
  - progii/tests/TestIntegridadPiezas.java: JUnit para probar que existen las clases, atributos y los métodos que representan a las piezas y que éstos tienen los nombres y los tipos correctos.
  - progii/tests/TestIntegridadGestorHistorial.java: JUnit para probar que existen las clases, atributos y los métodos del gestor del historial y que éstos tienen los nombres y los tipos correctos.
- Directorio partidas: contiene un conjunto de partidas guardas (no tienen por qué ser correctas desde el punto de vista del ajedrez) que sirven para cagar escenarios en las que probar el correcto funcionamiento de los desarrollos.

Los pasos a seguir para el desarrollo de la práctica serían:

1. Preparar un proyecto Eclipse con el contenido del fichero MiniAjedrez.zip.
2. Añadir al *Build Path* del proyecto todos los ficheros .jar contenidos en la carpeta lib.
3. El main para ejecutar el proyecto se encuentra en la clase `progii.juegotablero.gui.MainAjedrez`. Al ejecutarlo os aparecerá el error: java.lang.NoClassDefFoundError: progii/juegotablero/model/GestorHistorial. Para solucionarlo debéis crear la clase GestorHistorial en el paquete progii.juegotablero.model y los métodos (aunque no tengan código) descritos en el apartado Trabajo del Alumno.
4. Este mismo error se producirá con cada una de las piezas a implementar, así como con la clase `PiezaAjedrez`. Debéis crear todas las clases y las cabeceras de los métodos descritos en el apartado Trabajo del Alumno.
5. Las clases test.TestIntegridadGestorHistorial y test.TestIntegridadPiezas se encargan de comprobar que se han creado correctamente las clases y los métodos descritos en el apartado Trabajo del Alumno. Estos test no hacen ninguna prueba funcional de los métodos, únicamente comprueban que los nombres y los tipos son correctos. Hasta que estas pruebas no se ejecuten correctamente no funcionará el entorno de ventanas del ajedrez.
6. Una vez que hayáis conseguido que ambos test funcionen correctamente, ya podéis poneros a programar cada una de las clases que se piden en el apartado Trabajo del Alumno.

7. Para comprobar si los métodos de las piezas son correctos os recomendamos verificar las casillas que se ponen en rojo al seleccionar una pieza.
8. Para hacer algunas pruebas podéis usar las partidas guardas o crear otras nuevas.

4. Consideraciones de Entrega

Se entregarán los ficheros **PiezaAjedrez.java, Peon.java, Reina.java, Rey.java, Alfi.java, Caballo.java y GestorHistorial.java,** sin comprimir. Para que la entrega sea aceptada deberá superar todas las pruebas suministradas en los ficheros TestIntegridadPiezas, TestIntegridadGestorHistorial. Además de estas pruebas, el sistema de entrega llevará a cabo otra batería de pruebas para indicar la nota máxima a la que aspira el alumno y los errores detectados.

Al comienzo de cada fichero se debe rellenar el comentario con los nombres de los alumnos del grupo.

NOTA: Todos los ficheros deben comenzar con una anotación con los nombres de los alumnos del grupo:

```
@Programacion2 (
        nombreAutor1 = "nombre",
        apellidoAutor1 = "apellido1 apellido2",
        emailUPMAutor1 = "usr@alumnos.upm.es",
        nombreAutor2 = "",
        apellidoAutor2 = "",
        emailUPMAutor2 = ""
)
```

5. Errores graves a evitar

En esta sección se enumeran algunos errores que el alumno debe evitar. Esta lista no es exhaustiva en el sentido de que no recoge todos **los posibles errores que el alumno debe evitar cometer**.

1) Código innecesario o inalcanzable
2) Llamadas a métodos innecesarias o redundantes
3) Documentación deficiente (faltan comentarios significativos)
4) Atributos innecesarios (de clase o de instancia)
5) Identificadores no significativos
6) No sigue el convenio de nombres de Oracle
7) Código mal indentado
8) Uso innecesario de if para asignar o devolver valores booleanos
9) Se realizan operaciones de entrada/salida en alguna de las clases implementadas por el alumno, excepto en las ramas catch en donde sí que se permite.
10) Rama catch vacía

6. Pautas de programación a tener en cuenta

1. La utilización de nombres significativos para los identificadores, así como la utilización de los convenios de Java de Oracle.
2. La utilización de métodos auxiliares que implementen tareas comunes a varios métodos, y que de esa forma eviten la duplicidad de código.
3. Comentarios que describan cada clase y cada método.
4. El correcto indentado del código. Se recomienda la utilización en Eclipse de los atajos de teclado Crlt +i y Crtl+Shift+f.
5. La implementación de un código eficiente que no realice operaciones innecesarias.

7. Pruebas

La práctica será aceptada por el sistema de entrega, siempre y cuando supere las pruebas obligatorias del sistema de entrega. Los ficheros JUnit de la carpeta tests no realizan ninguna prueba del sistema, salvo TestTorre.java que prueba la torre, pero si vuestra práctica no supera correctamente estos test, el sistema de entrega no permitirá su entrega y gastaréis una entrega innecesariamente.