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 de la 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 posiblidad 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, se calificará como suspenso**. 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 puede 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 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, “comer al paso” del peón o la coronación del peón. Asimismo, la partida no termina con la existencia de un *jaque mate* (ni ésto 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: Juego que se ha de implementar

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 los movimientos válidos devueltos por la pieza. Para mover la pieza, el usuario deberá pinchar de nuevo sobre alguna de las casillas en rojo. Las casillas que se presentan en rojo no 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. Será al pinchar sobre la casilla roja a la que se quiere mover cuando se indique que el movimiento no se puede realizar, por ejemplo, porque atraviesa una pieza o bien porque se intentar mover a una casilla ocupada por una pieza de tu mismo color.

Las posiciones de las piezas se identifican de acuerdo a las posiciones de una matriz, la posición (0,0) es la que se encuentra más arriba a la izquierda y la posición (7,7) la posición más abajo a la derecha. En el historial de movimientos de la Figura 1 se ve como el peón blanco se ha movido de la posición (6,4) a la posición (4,4). Tal y como se ve en la Figura 1, las piezas negras empiezan siempre en la parte "alta" del tablero.

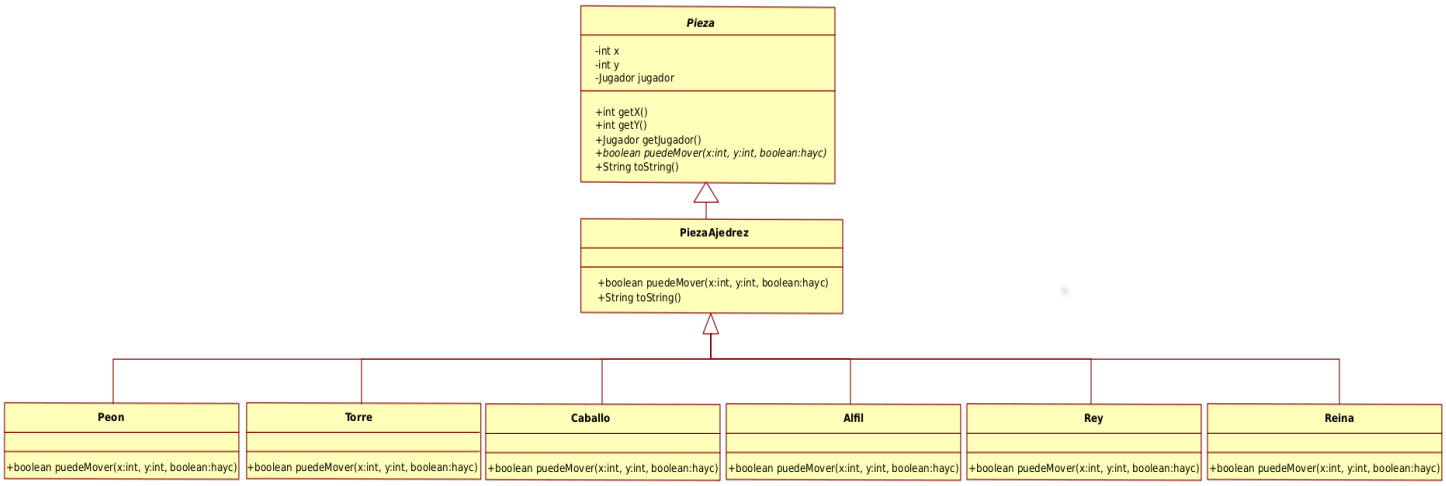
Para la implementación de este ejercicio se suministra el archivo **MiniAjedrez.jar** que contiene la implementación de una parte del juego. A continuación haremos una breve descripción de algunas de las clases que os resultarán útiles para el desarrollo de la práctica:

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 sobreescribir para conseguir el correcto funcionamiento del juego. La clase pieza dispone de los métodos `getX()` y `getY()` 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 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`.

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

El diagrama de clases que representa éstas clases es el siguiente:



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.

1.1. Trabajo del Alumno

El alumno debe implementar las siguientes clases:

- Clase **`PiezaAjedrez`**, que hereda de la clase Pieza y que debe estar en el paquete `progii.juegotablero.model.ajedrez`, y de la que se precisa:
  - Sobrescribir el método `puedeMover`. 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.
  - Sobrescribir el método `toString`. Este método debe devolver un String con el formato TipoPieza_NombreJugador. Para ello debéis utilizar los métodos `getTipoPieza()` y `getJugador().getNombre()` que se encuentran definidos en la clase Pieza y Pieza Ajedrez. Este método permitirá que las imágenes de las piezas se muestren correctamente por pantalla y se puedan

cargar/guardar partidas. Por ejemplo para la torre negra debería devolver TORRE_NEGRA.

- Las clases **`Alfil`**, **`Caballo`**, **`Rey`**, **`Reina`**, **`Peon`**, del paquete `progii.juegotablero.model.ajedrez.piezas` en las que se precisa:
  - Sobrescribir el método `puedeMover`. Este método debe implementar el movimiento de cada una de las piezas del ajedrez. 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 del peón depende del color del mismo, dicho color se puede consultar mediante el jugador al que pertenece la pieza con la siguiente comprobación:

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

También hay que tener en cuenta que el Peón es la única pieza cuyos movimientos dependen de si hay o no contrario en la casilla de destino.

- 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 GestorHisotial se precisa implementar:

  1. Contendrá 2 Atributos: `pilaDeshacer` y `pilaRehacer`, que son dos pilas de tipo `stacks.Stack` que almacenarán objetos de tipo `Movimiento` que se vayan produciendo en el desarrollo de la partida.
  2. El constructor sin parámetros que incializa las pilas `pilaDeshacer` y `pilaRehacer`.
  3. El método `guardarMovimiento`, que recibe como parámetro un `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 `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 `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 todas las librerías necesarias para el correcto funcionamiento de la práctica. Todos estos ficheros deben ser incluidos en el *Build Path* del proyecto para que todo funcione correctamente.
- 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 incomplet de la clase PiezaAjedrez.
  - progii/juegotablero/model/ajedrez/piezas/Torre.java: Clase que implementa la Torre y que podéis usar como inspiración.
  - 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

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 la 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 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 Trabajo del Alumno.
7. Como recomendación, el primer método a programar sería el toString () de la clase PiezaAjedrez, ya que permitirá que las imágenes de las piezas se muestren correctamente y así podáis empezar a programar más cómodamente los movimientos de las piezas.
8. Para comprobar si los métodos de las piezas son correctos os recomendamos utilizar las casillas que se ponen en rojo al seleccionar una pieza.
9. Para hacer algunas pruebas de las Piezas, podéis hacer un main creando una pieza y llamando al método puedeMover, por ejemplo para probar si se puede mover un peon negro, de (0,0) a (1,0) con contrario, podríamos hacer:

```
Peon p=new Peon(new Jugador ("", ControlJugadoresAjedrez.NEGRO),0,0);
System.out.println(p.puedeMover(1,0, true));
```

Este código también os podría valer para ver si una pieza se sale del tablero.

```
System.out.println(p.puedeMover(-1,8, true));
```

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.

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 Sun.

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 entregar. Los ficheros JUnit de la carpeta tests no realizan ninguna prueba del sistema, pero si vuestra práctica no supera correctamente estos test el sistema de entrega no permitirá su entrega y gastaréis una entrega innecesariamente.