CURSO 2024-25.
Práctica 4. Arrays y Ficheros.

1. Objetivos:

- Programar el juego de *MasterMind* mediante una aplicación basada en clases y objetos en Java.
- Manejar *arrays* correctamente cuando lo requiera la aplicación.
- Gestionar el manejo de ficheros para conseguir las funcionalidades de guardar la partida y de recuperar la partida.

2. El juego: el *MasterMind*.

Se trata de un juego de mesa clásico para dos jugadores, que se juega en un tablero. En nuestro caso, será un juego de un jugador contra el ordenador. Uno de los jugadores, en este caso el ordenador, pone una secuencia de fichas de colores, y la oculta al otro jugador. Este segundo jugador debe tratar de adivinar dicha secuencia de colores, para lo cual va proponiendo secuencias de fichas de colores. El primer jugador (ordenador) le va respondiendo pistas, indicativas de lo cerca o lo lejos que está de la secuencia oculta.

Las pistas que le va ofreciendo se basa en puntos negros y blancos:

- Un punto negro ● indica que ha acertado una ficha en color y posición.
- Un punto blanco ○ indica que ha acertado el color de una ficha, pero en una posición incorrecta.

Por ejemplo, supongamos que la secuencia oculta es: 🔴🟢🟡🔴

Si la jugada propuesta es 🔴🟣🟢🟡, las pistas serán ● ○ ○

- Un punto negro indicando que ha colocado una ficha correctamente, tanto en color como en posición (la primera ficha roja).
- Un punto blanco para indicar que la ficha verde existe en la combinación oculta, pero en una posición diferente.
- Un punto blanco para indicar que la ficha amarilla existe en la combinación oculta, pero en una posición diferente.

Se pueden repetir colores tanto en la secuencia oculta como en las jugadas propuestas.

A continuación, se muestra el resultado de una ejecución del juego. Para un mejor seguimiento, ten en cuenta que la combinación oculta es: 🟢🔴🔴🟡

```
¿Quieres recuperar una partida? (S/N): N
Número de fichas de las jugadas (4 - 6): 4

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): RRRR
Jugada 1    ■ ■ ■ ■   ● ●

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): RRAA
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): VVVV
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o
Jugada 3    ■ ■ ■ ■   ●

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): RRPP
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o
Jugada 3    ■ ■ ■ ■   ●
Jugada 4    ■ ■ ■ ■   ● o

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): RAVP
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o
Jugada 3    ■ ■ ■ ■   ●
Jugada 4    ■ ■ ■ ■   ● o
Jugada 5    ■ ■ ■ ■   o o o

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): ARRV
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o
Jugada 3    ■ ■ ■ ■   ●
Jugada 4    ■ ■ ■ ■   ● o
Jugada 5    ■ ■ ■ ■   o o o
Jugada 6    ■ ■ ■ ■   ● ● o o

Introduce jugada o G(guardar la partida).
R(Rojo), V(Verde), A(Amarillo), P(Púrpura): VRRA
Jugada 1    ■ ■ ■ ■   ● ●
Jugada 2    ■ ■ ■ ■   ● ● o
Jugada 3    ■ ■ ■ ■   ●
Jugada 4    ■ ■ ■ ■   ● o
Jugada 5    ■ ■ ■ ■   o o o
Jugada 6    ■ ■ ■ ■   ● ● o o
Jugada 7    ■ ■ ■ ■   ● ● ● ●
ACERTASTE LA JUGADA OCULTA
```

3. El programa a desarrollar.

Se desea escribir un programa que permita jugar a este juego. Descargue y descomprima el proyecto *IntelliJ* **FP Practica 4 - Arrays y Ficheros** de la plataforma Moodle, el cual contiene el código inicial de la práctica a realizar.

Una primera aproximación de las clases utilizadas es la siguiente:

1. **`MasterMind`**

   Es la clase principal que implementa la lógica del juego. La función *main* se encuentra aquí, y lo que hace es crear un objeto *MasterMind* y enviarle el mensaje *jugar()*.

   Los atributos que contiene la clase *MasterMind* son los siguientes

   - *jugadaOculta*: la jugada o secuencia oculta.
   - *tablero*: un tablero en el que se irán añadiendo las jugadas realizadas con sus pistas asociadas.
   - *numJugadas*: el tamaño de las jugadas, es decir, el número de fichas que componen cada jugada.

2. **`Jugada`**

   Se utiliza para representar a cada una de las jugadas realizadas en la partida. Una jugada está formada por una secuencia de colores, posiblemente repetidos. Esta clase sólo tiene un atributo de instancia:

   - *fichas*: *array* con los colores que definen la secuencia. Cada elemento es de tipo *Color*, que se ha definido como un enumerado con los valores ROJO, VERDE, AMARILLO y PÚRPURA.

   Existen además unas constantes para representar las jugadas con colores.

3. **`Pistas`**

   Define las pistas asociadas a una jugada. Dichas pistas consisten en el número de puntos negros y el número de puntos blancos. Precisamente esos son los atributos de la clase:

   - *aciertos*: el número de aciertos o puntos negros.
   - *descolocados*: El número de fichas descolocadas o puntos blancos.

   Existen además unas constantes para representar las pistas con puntos negros y blancos.

4. **`Tablero`**

   Define el tablero con todas las jugadas realizadas y sus pistas asociadas. El máximo número de jugadas que puede contener el tablero es 10. Si después de realizar 10 jugadas no se ha conseguido encontrar la secuencia oculta, el jugador pierde la partida.

   Los atributos de esta clase son:

   - *jugadas*: *array* con las jugadas.
   - *pistas*: *array* con las pistas asociadas a las jugadas.
   - *numJugadas*: número de jugadas que se llevan realizadas.

   Existe además una constante que indica el tamaño del tablero (máximo número de jugadas).

5. **`Teclado`**

   Contiene una serie de métodos de utilidad, todos ellos estáticos, para realizar la lectura por teclado de los datos que pide el juego.

A continuación, se detallan los métodos de cada una de las clases. Algunos de ellos ya están codificados en el enunciado, y **no podrán ser modificados en absoluto al realizar la práctica**. Otros vienen vacíos en el enunciado y tendrán que ser codificados para realizar la práctica (// TODO), pero **en ningún caso podrá modificarse la cabecera de dichos métodos**.

3.1. Clase *Teclado*:

Contiene los siguientes métodos **estáticos**, que son muy útiles para leer datos de la entrada:

- *int leerEntero(int menor, int mayor, int mensaje)*: lee de la consola un entero cuyo valor esté entre *menor* y *mayor*, y lo devuelve. El parámetro *mensaje* contiene el mensaje que visualiza el método para pedir el entero al usuario.
- *String leerString(String mensaje)*: lee una cadena de texto de la consola y la devuelve
- *char leerSiNo(String mensaje)*. Se utiliza cuando se realiza una pregunta con respuesta Sí o No (S/N). Devuelve el carácter 'S' o el carácter 'N'.
- *String leerJugadaGuardar(int longitud, String mensaje)*. Se utiliza cuando en el juego se pide al jugador que proponga una secuencia de colores de tamaño *longitud*. El jugador puede responder con la jugada (por ejemplo, RVVP) o bien, puede dar como respuesta G, si lo que quiere es guardar la partida.

No hay que implementar nada en esta clase.

3.2. Clase *Jugada*:

Contiene los siguientes métodos de instancia:

- *void visualizar():* sirve para visualizar una jugada. Por ejemplo, ■ ■ ■ ■.
- *String toString():* devuelve una cadena de caracteres con la representación de la jugada. Por ejemplo, RVVP.

Métodos que deben implementarse para hacer la práctica:

- Constructor *public Jugada(String cadena)*: a partir de una cadena de texto en donde se viene representada la jugada, por ejemplo, RVVP, debe inicializar el atributo *fichas*, formado por objetos de tipo *Color*. El tamaño del *array* viene determinado por el tamaño de *cadena*, y la asociación de colores: R es rojo, V es verde, A es amarillo y P es púrpura.
- Constructor *public Jugada(int numFichas)*: inicializa el atributo *fichas* con un *array* de objetos *Color*, donde el tamaño de dicho *array* es el indicado en *numFichas*, y cada uno de los colores se determina de forma aleatoria utilizando *Math.random()*. La idea es obtener un número aleatorio entre 0 y 3, de manera que el 0 sea rojo, el 1 verde, el 2 amarillo y el 3 púrpura.
- *public Pistas comprobar(Jugada oculta)*: en este método recibimos la jugada oculta (la que el jugador debe adivinar) y la comparamos con la jugada actual (*this*) para ofrecer las pistas oportunas (fichas negras y fichas blancas, o lo que es lo mismo, aciertos y descolocados).

  Para obtener adecuadamente las pistas hay que tener en cuenta que una ficha sólo puede servir para una pista como mucho. Nunca para dos o más. Por ejemplo, supongamos que la jugada actual es 🔴🟢🟢🟣 y la jugada oculta es 🔴🟡🟡🔴. No podemos dar como pistas una ficha colocada y otra descolocada ● ○, ya que estaríamos dando dos pistas basadas en la ficha roja de la jugada actual. Lo correcto sería decir que tenemos una ficha colocada (un punto negro).

  El mismo ejemplo lo podemos poner al revés, es decir, dar dos pistas por una ficha de la jugada oculta. Así, si la jugada actual es 🔴🔴🟢🟣 y la jugada oculta 🔴🟡🟡🟡, no podemos

dar como pistas un acierto y una descolocada. Lo correcto nuevamente sería decir que únicamente tenemos una ficha colocada (un punto negro).

Por tanto, se propone el siguiente algoritmo para implementar este método:

1. Calcular los aciertos o fichas colocadas. Para lo cual comparamos cada ficha de la jugada actual con la ficha de la misma posición en la jugada oculta. Cuando se encuentre un acierto debe marcarse la ficha tanto en la jugada actual como en la jugada oculta. Con ello conseguiremos que no se vuelta a dar ninguna pista utilizando ninguna de esas fichas. Para marcar las fichas ya utilizadas en alguna pista se recomienda utilizar sendos *arrays* de *booleanos*.
2. Calcular las fichas descolocadas: para cada ficha de la jugada actual que no esté marcada, comprobamos si esa ficha está en la jugada oculta no estando marcada. De igual manera, cuando se encuentre coincidencia de colores, tendremos que marcar la ficha tanto en la jugada actual como en la jugada oculta.

Una vez que tengamos el número de aciertos y el de descolocados, creamos el objeto *Pista* correspondiente y lo devolvemos.

3.3. Clase *Pistas*:

Contiene los siguientes métodos de instancia:

- Constructor que recibe el número de aciertos (fichas negras) y descolocados (fichas blancas), e inicializa los correspondientes atributos.
- *int getAciertos()*: devuelve el número de aciertos (fichas negras).
- *void visualizar()*: que sirve para visualizar las pistas con los puntos. Por ejemplo, ● ○ ○
- *String toString()*: devuelve una cadena de caracteres con la representación de las pistas. Por ejemplo, 1 2

No hay que implementar nada en esta clase.

3.4. Clase *Tablero*:

En esta clase hay que implementar todos sus métodos:

- Constructor *Tablero()*: debe inicializar los atributos correspondientes a un tablero al comienzo de una partida (no se han hecho jugadas). Cuidado porque hay que crear los *arrays* de jugadas y pistas. Aunque, lógicamente, quedarán vacíos.
- Los métodos getter *getNumJugadas*, *getJugadas* y *getPistas*.
- *void insertar(Jugada jugada, Pista pista)*: debe insertar una jugada junto con sus pistas en el tablero.
- *boolean completo()*: debe determinar si el tablero contiene el máximo de jugadas o no. Fíjese que la constante MAX_JUGADAS tiene el número máximo de jugadas que caben en un tablero.
- *void visualizar()*: debe visualizar el tablero en la pantalla, mostrando cada una de las jugadas realizadas con sus pistas asociadas. Para ello, este método utilizará los métodos *visualizar* de *Jugada* y de *Pista*. Por ejemplo, una visualización del tablero puede ser:

3.5. Clase *MasterMind*:

Esta clase tiene implementada la función *main*, en la que se pregunta al jugador si quiere recuperar una partida o si quiere empezar una partida nueva. En función de su respuesta, se crea un objeto *MasterMind* con un constructor u otro. Finalmente, se ejecuta el método *jugar* de dicho objeto *MasterMind*.

Métodos que deben implementarse para hacer la práctica:

- Constructor *Mastermind(int numFichas)*: inicializa el juego para una partida en la que todas las secuencias de colores tienen *numFichas* colores. Además del atributo *numFichas*, habrá que inicializar *jugadaOculta* con colores aleatorios y *tablero* con un tablero vacío.
- *void jugar()*: es el método que contiene la lógica del juego. Tendrá que ir pidiendo jugadas hasta finalizar el juego. Las jugadas se leen con *leerJugadaGuardar* de la clase *Teclado*, utilizando el mensaje "Introduce jugada o G (guardar la partida).\nR (Rojo), V(Verde), A (Amarillo), P (púrpura): ". La cadena que nos proporcione este método se procesará de la siguiente manera:
  - Si es "G", se pedirá el nombre del archivo con *leerString* de la clase *Teclado*, utilizando el mensaje "Nombre del archivo: ", se guardará la partida con el método *guardarPartida* y se terminará el juego.
  - En caso contrario (es una jugada), trataremos dicha jugada a través de un objeto de la clase *Jugada*. En primer lugar, la compararemos con la jugada oculta para obtener las pistas, introduciremos la jugada junto con sus pistas en el tablero, y seguidamente visualizaremos el tablero para que el jugador pueda ver el resultado.

    Si las pistas de la jugada indican pleno de aciertos, se sacará el mensaje "ACERTASTE LA JUGADA" y finalizaremos el juego.

    Si el tablero se ha llenado con esta jugada, se indicará con el mensaje "FIN DE LOS INTENTOS, NO CONSEGUISTE ACERTAR", se visualizará la jugada oculta y finalizaremos el juego.

  Una vez terminado el método *jugar*, ya puedes probar tu práctica de *MasterMind* jugando tus primeras partidas. La única funcionalidad que le falta es guardar y recuperar partidas.
- *void guardarPartida(String nombreArchivo)*: este método se utiliza para guardar el estado de la partida en un archivo de texto. La jugada oculta se guardará en la primera línea del archivo y, a continuación, una línea para cada una de las jugadas realizadas por el jugador, junto con su pista asociada.

  Por ejemplo, sea fichero *prueba.txt* obtenido con este método:

  ```
  VRPR
  RRRR 2 0
  VVVV 1 0
  AAAA 0 0
  PPPP 1 0
  ```

  Debe cerrarse el fichero antes de terminar el método. En caso de que haya algún problema con el fichero, se visualizará el mensaje "ERROR AL GUARDAR LA PARTIDA".
- Constructor *MasterMind(String nombreArchivo)*: debe inicializar los atributos de un objeto *MasterMind* utilizando el contenido de un archivo de texto en el que previamente se haya guardado una partida. Con la primera línea del fichero podemos dar valores iniciales a *numFichas* y a *jugadaOculta*. Por otra parte, *tablero* será inicialmente un tablero vacío, pero iremos insertándole las jugadas y pistas contenidas en cada una de las líneas del fichero.

Debe cerrarse el fichero antes de terminar el método. En caso de que haya algún problema con el fichero de entrada, se visualizará el mensaje "ERROR AL RECUPERAR LA PARTIDA".

Ahora ya puedes probar la funcionalidad completa del juego. Prueba a guardar una partida a medias y a recuperarla posteriormente para poder terminarla.

4. Entrega de la práctica.

Se entregará el proyecto *IntelliJ* resultante de hacer la práctica, comprimiéndolo previamente en un archivo ZIP. No importa si contiene o no archivos con partidas a medias. El nombre del archivo será el mismo que tenía el proyecto inicial: "**FP Practica 4 - Arrays y Ficheros.zip**", y se subirá a la plataforma dentro del plazo de entrega.