PROYECTO MASTERMIND 2.0

CONTENIDO

INTRODUCCIÓN

"Cáete siete veces y levántate ocho."

Proverbio japonés

El entrenamiento continúa, y el proyecto **MasterMind** también. Esta vez lo único que haremos será cambiar el funcionamiento del juego para que admita letras repetidas en los códigos. Este aspecto, en el proyecto anterior tenía como dificultad controlar la generación de letras en el arreglo para que no hubiera repetidas; esto ahora es más fácil porque no hay que controlar nada, solo generar letras aleatorias.

Sin embargo, ahora se complicará bastante la corrección de códigos y el cálculo de las notas, ya que los regulares pueden resultar confusos, incluso cuando uno lo mira en una hoja de papel.

OBJETIVO

Crear un programa en Java que implemente el juego de lógica **MasterMind** tal como será descrito en este documento. El estudiante deberá poder implementar el programa en cuestión haciendo uso de todas las herramientas vistas hasta ahora en el curso, pudiendo consultar a los docentes a través de la plataforma (***Udemy*** o ***KA EduSoft***).

Este proyecto tiene como objetivo afianzar todos los conceptos que el estudiante ya aprendió en las clases, enfatizando el uso de arreglos, condiciones y repetición para crear un programa con amplias posibilidades.

EL MASTERMIND

El juego **MasterMind** es ampliamente conocido, existiendo su versión digital (programa o videojuego) y su versión de mesa, tal como se ve en la imagen de abajo:

El juego es para dos jugadores, uno que cumple el rol de **Pensador** y otro el rol de **Adivinador**.

En el juego original el **Pensador** justamente piensa una secuencia de colores (pueden ser repetidos o no) y el **Adivinador** intentará descubrir cual es dicha secuencia.

Para lograr lo anterior, el **Adivinador** le irá presentando al Pensador diferentes secuencias de colores y el **Pensador** le ira dando pistas: cada color que el **Adivinador** ha colocado en un lugar correcto lo marcará como bueno, y cada color que el **Adivinador** ha ubicado en un lugar incorrecto la marcará como regular. Los colores que el **Adivinador** ha puesto en su secuencia y que no están en la secuencia del **Pensador** no tendrán ninguna evaluación.

Puedes ver una versión Web de esta mecánica de juego simulando al juego de mesa haciendo clic aquí.

El Adivinador tendrá una cantidad finita de intentos para intentar adivinar el código del **Pensador**. En cada intento el **Adivinador** corregirá sus secuencias siguiendo las pisas del **Pensador** acercándose poco a poco a la secuencia que éste ha ideado.

Si el **Adivinador** consume todos los intentos sin adivinar pues habrá perdido y el **Pensador** habrá ganado. Si el **Adivinador** adivina el código antes de agotar sus intentos pues habrá ganado y el **Pensador** habrá perdido.

MASTERMIND: VERSIÓN DE ESTE PROYECTO

En este proyecto programaremos dicho juego, pero utilizando letras en vez de colores, y cambiando levemente las reglas de las pistas que se le dan al **Adivinador**.

El **Pensador** ideará un código de **4 letras**, en esta versión mayúsculas entre la A y la H, las cuales pueden estar repetidas. Por ejemplo: ABBD. El **Adivinador** mostrará diferentes códigos teniendo un máximo de 10 intentos para adivinar. Cada vez que el **Adivinador** presenta un código, el **Pensador** mostrará dos notas:

- Buenos: para indicar la cantidad de letras que el **Adivinador** ha acertado en su código y que están en la ubicación correcta con respecto al código del **Pensador**.
- Regulares: para indicar la cantidad de letras que el **Adivinado** ha acertado en su código y que están en una ubicación diferente con respecto al código del **Pensador**.

Por ejemplo, si el Pensador ha idea el código tal ABBD, y el Adivinador presenta el código EBFG, tenemos que la letra **B** en su código existe también en el código del **Pensador**, y además está en la posición indicada, así que es **1 bueno**. Las demás letras no están en el código del **Pensador**, así que hay **0 regulares**.

Imagina ahora que el **Adivinador** presenta el código EABB. Pues la letra **A** está en el código del **Pensador**, pero en una posición diferente, por lo cual es **1 regular**. Luego tenemos que la letra **B** en el código del **Adivinador** está en una posición correcta (letra en el tercer lugar) que es **1 bueno**; el asunto con la cuarta letra **B** en el código del **Adivinador** porque podría ser confundida con un regular, pero no lo es porque esa posición ya fue evaluada como buena.

Este aspecto es de suma importancia. Si el **Adivinador** presenta, por ejemplo, el código BGBB ¿cuáles son las notas? Pues tenemos que la primera **B** podría ser un regular con respecto a la **B** en el código del pensador (ABBD), sin embargo, eso estaría mal, porque en el código del **Adivinador** hay una **B** en la tercera posición coincidiendo con la tercera **B** del **Pensador**, así que en realidad es **1 bueno**. Hay un regular al comparar la primera **B** del **Adivinador** con la segunda **B** del **Pensador**. Luego, la tercera **B** del **Adivinador** ya no tiene con quién ser comparada (todas las letras han sido evaluadas). Las notas finales son **1 bueno** y **1 regular**.

Otro ejemplo podría ser que el Adivinador presente el código BBBB ¿cuáles son las notas? Pues **2 buenos** y **0 regulares**.

EL PROGRAMA

En esta versión del programa el **Adivinador** será el usuario y el **Pensador** la CPU (nuestro programa). El sistema generará un código al azar con letras que pueden estar repetidas (dependerá del azar), el cual no será mostrado al usuario. Luego el usuario ingresará códigos posibles y el programa le mostrara las notas de su código con respecto al código secreto, tal como se ha descrito anteriormente.

Si el usuario (**Adivinador**) descubre el código secreto antes de agotar los intentos se le indicará que ha ganado, en caso contrario se le indicará que ha perdido y cuál era el código que debía adivinar.

ESPECIFICACIÓN

Se pide implementar un programa en Java que haga exactamente lo que se acaba de describir. Básicamente funciona todo igual que en el proyecto anterior, simplemente has de modificar cómo se generan los códigos al azar para que se admitan letras repetidas, y cómo corriges las notas.

IMPLEMENTACIÓN

Para llevar adelante este proyecto deberás utilizar los siguientes tipos que te damos ya declarados:

```
final byte MAX_INTENTOS = 10;
final byte LARGO_CODIGO = 4;
final char PRIMERA_LETRA = 'A';
final char ULTIMA_LETRA = 'H';
```

No hay cambios con respecto a los requerimientos de la versión anterior. Como ya se dijo, solo tienes que cambiar cómo se generan los códigos aleatorios en tu programa y cómo se corrigen las notas.

GENERACIÓN DE LETRAS AL AZAR

Este aspecto ya fue explicado en el proyecto anterior y ya lo tienes programado. No es necesario hacer nada en este aspecto. Reutiliza tu propio código fuente.

CORRECCIÓN DE NOTAS

A continuación presentamos un posible algoritmo para la corrección de notas. Puedes intentar seguirlo en tu programa o bien hacerlo a tu manera.

La dificultad en la corrección de notas radica en la repetición de letras, ya que se puede tener más regulares de los debidos.

El secreto está en evaluar primero lo buenos, y luego ya no utilizar las posiciones evaluadas en el código para calcular los regulares. Por ejemplo, sea el código del **Pensador** BBBD, y el **Adivinador** presenta CBBB.

El procedimiento sería, evaluar primero los **buenos** (si los hubiera), y luego los **regulares**. En este caso vemos que las posiciones 2 y 3 de ambos códigos coinciden, así que tenemos **2 buenos**. Marcaremos de verde esto para que se visualice: **Pensador** → BBBD, **Adivinador** → CBBB.

Ahora tenemos que evaluar la letra C, y como no está en el código del Pensador pues la marcamos también como evaluada y no sumamos nada a ninguna nota: **Pensador** → BBBD, **Adivinador** → CBBB.

Finalmente tenemos la última **B** del **Adivinador**, y dos posiciones disponibles aún en el **Pensador**. Así que vemos que la letra **B** final es un regular con respecto a la primera letra **B** del código del **Pensador**. Esto se dificultaría aún más en códigos de más de 4 letras. Así pues tenemos **2 buenos** y **1 regular**.

Teniendo en cuenta esto, y puesto que se pide que tu código funcione con cualquier valor de `LARGO_CODIGO` así como cualquier rango de caracteres según se establezca en `PRIMERA_LETRA` y `ULTIMA_LETRA`, te mostramos algunos ejemplos de lo que te podrías encontrar:

Código del Pensador: ABACB		
Código Adivinador	**Buenos**	**Regulares**
ABBBA	2	2
AAAAA	2	0
CACAA	0	3
CCCBB	1	2

SUGERENCIA DE ALGORITMO

Declarar dos arreglos anónimos de tipo **boolean**:

```
boolean[] evaluadasP= new boolean[LARGO_CODIGO],
          evaluadasA= new boolean[LARGO_CODIGO];
```

Estos arreglos iniciarán todas sus celdas en `false`. Luego se haría lo siguiente:

- Recorrer ambos códigos mirando las mismas posiciones:
  - Si dos letras en una misma posición son iguales:
    - Sumar 1 **bueno**.
    - Poner en **true** `evaluadasP` y `evaluadasA` en las mismas posiciones.
- Recorrer el arreglo del **Adivinador** hasta el final:
  - En cada posición no evaluada el **Adivinador** recorrer todo el código del **Pensador**:
    - Si la celda del **Pensador** no fue evaluada y es igual a la del **Adivinador**:
      - Sumar 1 **regular**.
      - Poner en `TRUE evaluadasP` en la posición del **Pensador**.
      - Poner en `TRUE evaluadasA` en la posición del **Adivinador**.

Tienes la libertad de utilizar el algoritmo que quieras y modificar este a tu antojo.

RESTRICCIONES DE CÓDIGO

Para este proyecto tienes que utilizar las constantes definidas arriba tal cual se te han indicado (básicamente copia y pega). Además tienes las siguientes restricciones:

- No puedes declararte ningún otro tipo de datos, salvo de forma anónima como se mostró.
- No puedes declarar nuevas constantes.
- Debes utilizar las constantes ya definidas para tus bloques **FOR**, **WHILE** o **REPEAT** que utilices para recorrer arreglos.

Puedes declararte tantas variables como quieras y utilizar el algoritmo que quieras. Cualquier otra facilidad de Pascal o de Lazarus no será admitida en este proyecto.

SE PIDE

Escribir un programa Java que cumpla con todo lo descrito en el documento y funcione exactamente igual que los ejemplos. Se te proveerá del archivo **.jar** ya compilado para que puedas probarlo en funcionamiento y ver lo que tienes que lograr.

ENTREGA DEL PROYECTO

Este proyecto es obligatorio y debe ser entregado ya que será evaluado por un docente. Lo que debes entregar es el archivo **.java** con el código fuente y el archivo **.jar** que hayas generado. Dependiendo de la plataforma en que hagas el curso será el medio por el cual enviarás el archivo:

- KA EduSoft: En la lección donde se te presenta este proyecto tendrás la opción para subir el archivo del código fuente.
- Udemy: Escribe un correo electrónico a bedelia@kaedusoft.edu.uy con el asunto **PROYECTO JAVA MasterMind2.0 [NOMBRE] [APELLIDO]** en el cual adjuntarás el archivo con el código fuente de tu trabajo.

Si tienes alguna duda escribe a bedelia@kaedusoft.edu.uy.