PROYECTO MASTERMIND 1.0

CONTENIDO

INTRODUCCIÓN

"No importa lo que hagas en la vida, hazlo con todo tu corazón."

Confucio

Como ya te habrás dado cuenta, los proyectos tienen una dificultad progresiva, es decir, cada uno implica un reto mayor que el anterior, permitiéndote hacer uso de las habilidades mentales que ya desarrollaste (tanto en tu vida como en lo que ya va del curso) y desarrollar nuevas.

Al plantearte proyectos retadores te permitimos exprimir tus recursos actuales y así forjar nuevos. Esa es la clave. Además, la repetición y el uso de todo lo ya aprendido hará que realmente comprendas cómo funciona todo y te conviertas en un/a verdadero/a programador/a.

OBJETIVO

Crear un programa en Java que implemente el juego de lógica MasterMind tal como será descrito en este documento. El/La 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/la 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 pistas 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 no pueden estar repetidas. Por ejemplo: ABCD. 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 que oficiarán de pistas:

- 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 ideado el código ABCD, 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 EAFG. Pues la letra **A** está en el código del **Pensador**, pero en una posición diferente, por lo cual es **1 regular**. Ninguna otra letra está en el código del **Pensador** así que no hay más notas que asignar, teniendo **0 buenos** y **1 regular** como resultado final.

Otros ejemplos podrían ser:

Código del Pensador: ABCD		
Código del Adivinador	**Buenos**	**Regulares**
EFGH	**0**	**0**
EFAG	**0**	**1** (la A)
EBFG	**1** (la B)	**0**
ABFG	**2** (la A y la B)	**0**
ABFC	**2** (la A y la B)	**1** (la C)
ABDC	**2** (la A y la B)	**2** (la C y la D)

Es importante notar que en esta versión las notas no dicen cuál letra es la que asigna dicha nota. En la tabla lo hemos agregado para que puedas entender el mecanismo, pero el **Pensador** únicamente mostrará el número de **Buenos** y el número de **Regulares** al **Adivinador**. Esto le añade una dificultad extra al juego.

En esta versión el **Adivinador** tendrá un máximo de **10 intentos** para adivinar el código del **Pensador**, tras lo cual, si no descubre la secuencia de letras, habrá perdido. Resumiendo entonces:

- ✓ Códigos de **4 letras** de la **A** a la **H**.
- ✓ No se pueden repetir letras.
- ✓ El **Adivinador** tiene **10 intentos**.
- ✓ El **Pensador** indica dos notas: **Buenos** y **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 no repetidas, 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 Pascal que haga exactamente lo que se acaba de describir. Veamos un ejemplo de ejecución para que puedas ver exactamente lo que debes lograr:

```
MasterMind V1.0
Dispones de 10 intentos para adivinar el codigo.
Codigo 1 de 10>>
```

Como ves, lo primero que se muestra es el nombre del programa y la versión, lo cual también ha de aparecer en tu proyecto. Luego se indica al usuario la cantidad de intentos que posee para adivinar el código, tras lo cual se muestra el PROMPT o espera del programa para que el usuario ingrese un código. El PROMPT indica el número de intento actual (en este caso **1**).

A continuación, mostraremos en color **verde** la entrada del usuario para que puedas distinguirla, pero obviamente en tu programa ese color no estará (ya lo has visto en el video en mostramos el funcionamiento, además de que podrás descargarte una versión funcional del mismo). Supongamos que el **Adivinador** (tú) ingresa el código **ABCD**:

```
MasterMind V1.0
Dispones de 10 intentos para adivinar el codigo.
Codigo 1 de 10>> ABCD
B= 2 R= 0
Codigo 2 de 10>>
```

El **Pensador** (nuestro programa) ha mostrado las notas que indican **2 buenos** y **0 regulares**. Así que sabemos que hay dos letras correctas en el código. Lugo se muestra nuevamente el PROMPT que indica el número de intento actual. Ingresaremos entonces un nuevo código, **BFCE**:

```
MasterMind V1.0
Dispones de 10 intentos para adivinar el codigo.
Codigo 1 de 10>> ABCD
B= 2 R= 0
Codigo 2 de 10>> BFCE
B= 1 R= 2
Codigo 3 de 10>>
```

Ahora se le indica al usuario que tiene **1 bueno** y **2 regulares**. Como ves, no se sabe exactamente cuál letra es la que es correcta, cuál está en una posición diferente y cuál no está en el código secreto, eso es algo que ha de deducirse de forma lógica.

Cuando el jugador acierta el código secreto, el programa le mostrará un mensaje indicándole que ha ganado y finalizará su ejecución, tal como se muestra a continuación:

```
MasterMind V1.0
Dispones de 10 intentos para adivinar el codigo.
Codigo 1 de 10>> ABCD
B= 2 R= 0
Codigo 2 de 10>> BFCE
B= 1 R= 2
Codigo 3 de 10>> FBCH
B= 4 R= 0
EXCELENTE!!! Ganaste.
```

Cuando el jugador agota todos los intentos sin adivinar el código secreto, el programa indicará dicho suceso, mostrará el código que debía adivinarse y terminará su ejecución:

```
MasterMind V1.0
Dispones de 10 intentos para adivinar el codigo.
Codigo 1 de 10>> ABCD
B= 2 R= 0
Codigo 2 de 10>> BFCE
B= 1 R= 2
Codigo 3 de 10>> ACBD
B= 0 R= 2
Codigo 4 de 10>> HBCF
B= 2 R= 2
Codigo 5 de 10>> GEDC
B= 0 R= 1
Codigo 6 de 10>> FBGD
B= 2 R= 0
Codigo 7 de 10>> BACD
B= 0 R= 2
Codigo 8 de 10>> ABFH
B= 2 R= 1
Codigo 9 de 10>> FBCH
B= 4 R= 0
Codigo 10 de 10>> HCBF
B= 0 R= 4
PERDISTE!!! El código era FBCH
```

Este es el comportamiento que deberás lograr programar, teniendo en cuenta que:

- Se asume que el usuario siempre ingresa la cantidad de letras adecuadas.
- Se asume que el usuario siempre ingresa letras mayúsculas.
- Se asume que el usuario siempre ingresa letras en el rango correcto.
- **Dicho de otro modo: el programa no debe preocuparse por capturar errores en la entrada ya que el usuario siempre ingresará una entrada correcta.**

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';
```

Tu código debe utilizar estas constantes en su implementación. No puedes declararte otros tipos de datos (clases) ni otras constantes. Tu código debe estar programado de tal modo que si se cambia el valor de alguna constante todo funcione correctamente. Por ejemplo, si se cambia el valor de **MAX_INTENTOS** el programa debe funcionar con dicho valor. Si se cambia el valor de **PRIMERA_LETRA** y/o de **ULTIMA_LETRA** el programa debe admitir códigos entre dicho rango de letras.

GENERACIÓN DE LETRAS AL AZAR

La función **nextInt(n)** de la clase **Random** sirve para obtener números aleatorios entre `0` y un `n-1` tal como ya hemos visto a lo largo del curso, pero no sirve para generar caracteres al azar. Sin embargo, sabiendo que los caracteres están ordenados en la tabla ASCII según un valor numérico (su índice), es posible generar un número al azar que coincida con la letra que queremos obtener y luego utilizar el casteo `(char)` para obtener la letra en cuestión.

En concreto nuestras letras van desde la **A** hasta la **H**, es decir desde el índice **65** de la tabla hasta el índice **72** de la tabla. La gracia está entonces en poder generar números al azar entre 65 y 72 inclusive. Esto ya lo hemos visto en clases anteriores. En concreto en este caso sería:

```
numero= generador.nextInt(72-65+1)+65;
```

En este ejemplo, **generador** es un objeto (variable) de tipo `Random`, tal como ya se ha hecho en clases previas a este proyecto, con la cual se utiliza la función **nextInt** para obtener un número aleatorio. De forma genérica, si tenemos un número X y otro número Y que es mayor, y queremos generar números aleatorios entre X e Y, se debería utilizar **nextInt** de la siguiente manera:

```
numero= generador.nextInt(Y-X+1)+X;
```

Ahora bien, tal como hemos enseñado en el curso, es aconsejable utilizar constantes para nuestros programas, y como ya hemos definido 2 en este proyecto, las usaremos para esta tarea. Por tanto, la fórmula final queda así:

```
indice= generador.nextInt((int)ULTIMA_LETRA-(int)PRIMERA_LETRA)+1+(int)(PRIMERA_LETRA);
```

De este modo ya tienes la fórmula que te da como resultado un índice ASCII entre 65 y 72 pero utilizando las constantes definidas. Así, al cambiar algún valor de estas constantes el programa igualmente funcionará de manera correcta. Lo que te queda por hacer es simplemente obtener el carácter del índice generado utilizando `(char)`.

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, clases o componentes extra, salvo variables.
- No puedes declarar nuevas constantes.
- Debes utilizar las constantes ya definidas para tus bloques **FOR**, **WHILE** o **REPEAT** que utilices para recorrer arreglos.

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 probar este programa 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 también tu archivo **.jar** para comprobar que puedes utilizar correctamente el código brindado para **pom.xml**. 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 MasterMind1.0 [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.