PROYECTO MASTERMIND 2.5

CONTENIDO

INTRODUCCIÓN

"Todo comienza siendo difícil y luego de practicarlo termina siendo fácil. Eso es lo que hace que el aprendizaje sea un regalo, porque lo que aprendes al final te pertenece."

Anónimo

Seguiremos trabajando con el proyecto **MasterMind** mejorándolo cada vez más. En este caso el proyecto no variará demasiado en su funcionamiento, pero sí en su implementación, ya que será necesario reescribirlo para que funcione mediante subprogramas.

Esto te permitirá aprender a dividir un problema grande en problemas más pequeños (cuestión que ya estás haciendo desde que empezó el curso), definiendo tareas específicas para cada bloque de código.

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 PROGRAMA

En esta versión, en esencia, el programa funcionará igual que la versión anterior, salvo que se incluirá la captura de errores: si el usuario ingresa un carácter que no esté entre `PRIMERA_LETRA` y `ULTIMA_LETRA`, o bien, si se ingresa un número de letras diferente de `LARGO_CODIGO`, se mostrará el mensaje:

```
ERROR: El código no es valido. Ingresa otro con LARGO_CODIGO letras entre
PRIMERA_LETRA y ULTIMA_LETRA>>
```

Obviamente en la salida se muestran los valores de las constantes. Hecho se eso el programa quedará a la espera de que se ingrese un valor correcto, o seguirá mostrando el mensaje cada vez que el usuario ingrese algo no válido.

ESPECIFICACIÓN

El programa funcionará de la misma manera que la versión anterior, mostrando la versión del mismo y dejando el prompt a la espera de que el usuario ingrese un código:

```
MasterMind V2.5
Dispones de 10 para adivinar el codigo.
Codigo 1 de 10>>
```

La única variante surge por la captura de errores, cuando el usuario ingresa algo que no corresponde a lo esperado (una letra fuera de rango o un largo de código incorrecto). Por ejemplo, imaginemos que el usuario ingresa el código **`ATEE`** (la letra T está fuera de rango):

```
MasterMind V2.5
Dispones de 10 para adivinar el codigo.
Codigo 1 de 10>> ATEE
ERROR: El codigo no es valido. Ingresa otro con 4 letras entre A y H>>
```

Como ves, se muestra el mensaje y se queda a la espera de un nuevo código. No se aumenta el número de intento ni se evalúa nada. Imagina que el usuario ahora ingresa el código **AE** (solo dos letras):

```
Codigo 1 de 10>> ATEE
ERROR: El codigo no es valido. Ingresa otro con 4 letras entre A y H>>AE
ERROR: El codigo no es valido. Ingresa otro con 4 letras entre A y H>>
```

El programa seguirá pidiendo un código válido hasta obtener uno y no avanzará mientras eso no suceda. Por lo demás, todo funcionará de manera habitual. Si el usuario ahora ingresara AAEE:

```
Codigo 1 de 10>> ATEE
ERROR: El codigo no es valido. Ingresa otro con 4 letras entre A y H>>AE
ERROR: El codigo no es valido. Ingresa otro con 4 letras entre A y H>>AAEE
B= 1 R= 1
Codigo 2 de 10>>
```

El programa muestra las notas y continúa con su ejecución normal.

IMPLEMENTACIÓN

Para llevar adelante este proyecto deberás utilizar los tipos de datos de las versiones anteriores:

```
public static final int MAX_INTENTOS = 10;
public static final int LARGO_CODIGO = 4;
public static final char PRIMERA_LETRA = 'A';
public static final char ULTIMA_LETRA = 'H';
```

Ahora bien, el cambio drástico viene porque has de implementar los siguientes cuatro subprogramas:

```
/**
 * Genera un código al azar y lo asigna a la variable codigo. El codigo generado
 * puede contener letras repetidas.
 * @return Un código de LARGO_CODIGO caracteres generados aleatoriamente.
 */
public static char[] generarCodigo(){

}


/** Lee el codigo de la entrada estandar y lo asigna a la variable codigo. Ademas
* retorna el valor TRUE si el codigo leido es correcto, FALSE si no.
* El codigo leido puede ser incorrecto si:<br>
* <ul><li>Contiene uno o más caracteres fuera del rango [PRIMERA_LETRA,ULTIMA_LETRA].</li>
*  <li>No contiene el largo LARGO_CODIGO</li>
* </ul>
 * @param codigo Contendrá el código leído desde la entrada. Si hay un error
 * no se garantiza la consistencia del mismo. Verificar, para ello, el retorno
 * de esta operación.
 * @return true si el código cumple con las condiciones, false en caso contrario.
 */
public static boolean leerCodigo(char[] codigo){

}

/**
 * Imprime en la salida estándar el código pasado como argumento. Deja el
 * cursor al final de la impresión si generar una nueva línea.
 * @param codigo El código que se desea imprimir.
 */
```

```
public static void imprimirCodigo(char[] codigo){

}

/**
 * Calcula las notas de <b>codAdivinador</b> en función de <b>codPensador</b>. Asigna los
buenos
 * y los regulares a los argumentos con el mismo nombre
 * @param codAdivinador El código del adivinador. Se asume que es un código correcto.
 * @param codPensador El código del pensador. Se asume que es un código correcto.
 * @param buenos El cálculo de buenos será asignado a este parámetro.
 * @param regulares El cálculo de regulares será asignado a este parámetro.
 */
public static void calcularNota(char[] codAdivinador, char[] codPensador, AtomicInteger
buenos, AtomicInteger regulares){

}
```

Estos subprogramas deben funcionar tal como lo indican los comentarios que están encima de ellos. Luego, tu programa principal (método **main**) ha de utilizarlos para hacer funcionar el juego tal como ya lo hacía en versiones anteriores.

RESTRICCIONES DE CÓDIGO

Para este proyecto tienes que utilizar las constantes definidas arriba tal cual se te han indicado (básicamente copia y pega), implementando y utilizando los suprogramas dados anteriormente tal cual han sido presentados (sus firmas deben ser idénticas). Además tienes las siguientes restricciones:

- No puedes declararte ningún otro tipo de datos globales.
- Puedes declararte cualquier constante o variable dentro de los subprogramas.
- Puedes declarar tantas variables globales como quieras.
- Puedes crear nuevos procedimientos y funciones para auxiliarte si lo necesitas, pero éstos no pueden ser usados en el procedimiento **main**, ya que en él solo se admiten los subprogramas dados en esta letra.
- No puedes declarar nuevas constantes globales.
- Debes utilizar las constantes ya definidas para tus bloques **`FOR`**, **`WHILE`** o **`DO..WHILE`** que utilices para recorrer arreglos.

Cualquier otra facilidad de Java o de NetBeans 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** para que puedas probar la ejecución del programa.

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 con el código fuente. Según donde haces el curso, la entrega se efectúa así:

- 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 SE MasterMind2.5 [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 o al WhtasApp o Telegram **+598 94 815 035**.