

# **Práctica 1**

## **GUI (Interfaz Gráfica de Usuario)**

Programación Orientada a Objetos

Humberto Alejandro Ortega Alcocer (2016630495)

Opción: 9 “Juego de Adivinar un Número”, inciso b.

6 de noviembre de 2020

Grupo: 2CM1

# Introducción

El juego “Adivina el Número”, es un juego que se fundamenta en un conocido algoritmo conocido como *Búsqueda Binaria*, en dicho algoritmo, se realiza una búsqueda considerando que los elementos se encuentran previamente ordenados por lo que podemos determinar, dado un elemento en cuestión, si hemos encontrado el valor que buscábamos, si el valor que buscábamos es menor al punto dónde nos encontramos en la búsqueda o si el valor que buscábamos es mayor al punto dónde nos encontramos en la búsqueda. Dicho algoritmo, es extremadamente eficiente ya que, en cada paso de ejecución del mismo, podemos descartar la mitad restante del conjunto de datos a analizar.

Para este juego, el usuario es nuestro “algoritmo”, en el sentido de que, la opción más hábil para jugar, será siempre encontrar la mitad inferior o superior inmediata al valor ingresado para poder encontrar el *número oculto* lo más rápido posible.

La ejecución del juego es muy simple, iniciamos con un *número oculto* el cual estará dentro del rango 1 - 100, y el usuario tendrá 6 intentos para poder adivinarlo. El programa, a su vez, presentará al usuario un mensaje que indique si el valor que ingresó el usuario es mayor, menor, o igual al número oculto.

Una vez agotados los intentos disponibles, el programa muestra el número oculto y se deberá ejecutar de nuevo para poder jugar.

El planteamiento de nuestro juego será, en palabras del profesor Roberto Tecla:

## 9.-Juego de Adivinar un numero

- a) El programa genera aleatoriamente un numero comprendido entre 1 y 1000. Si el usuario acierta el numero gana si no el programa ajusta el intervalo basado en la respuesta del usuario y permite que el usuario intente de nuevo.
- b) El programa genera aleatoriamente un numero comprendido entre 1 y 100 que el usuario tiene que adivinar en un máximo de 6 intentos. Si el usuario escribe en un campo de texto su numero y presiona un botón entonces el programa le “informara” en una etiqueta si el numero generado es mayor, menor, o igual que el numero escrito en el campo de texto. El juego termina si el usuario adivina el numero o si se le terminan las oportunidades (pueden ser 6).

## Desarrollo

Para el desarrollo del juego, comenzamos definiendo el diagrama UML de cada una de las clases que estarán interactuando entre sí. Es importante que separemos nuestra implementación de interfaz gráfica (GUI) de nuestra implementación lógica. Es por ello que consideramos las siguientes clases:

- AdivinaNumero
- AdivinaNumeroGUI
- Main

En donde cada una de estas clases realiza tareas muy particulares y definidas que describiremos a continuación.

Dentro de la clase **AdivinaNumero** colocaremos toda la lógica correspondiente a nuestro juego que, en realidad, no es mucha pero requiere también de la implementación de un par de funciones que nos permitan, desde la clase encargada de procesar la *interfaz gráfica de usuario* (**AdivinaNumeroGUI**), determinar el estado del juego así como obtener datos simples para poder mostrar el número de intentos restantes, si ya hay un ganador y el número secreto.

Lo primero que debemos resolver, es generar el número aleatorio, el cual generaremos en el constructor de la clase, junto a la inicialización de algunas variables miembro para el estado inicial del juego:

```
public AdivinaNumero() {  
    int limiteSuperior = 100; // Variable para determinar el límite superior para el número aleatorio a generar.  
    int limiteInferior = 1; // Variable para determinar el límite inferior para el número aleatorio a generar.  
    this.numero = (int)(Math.random() * (limiteSuperior - limiteInferior + 1) + limiteInferior);  
    // Generamos y asignamos un número aleatorio en el rango [limiteInferior - limiteSuperior]  
  
    this.intentos = 6; // 6 intentos al inicio.  
    this.hayGanador = false; // Iniciamos sin ganador.  
}
```

Algunos detalles a mencionar es que he definido el límite superior para el número aleatorio a ser generado en 100 y el límite inferior en 1, teniendo un rango de 1 a 100 con valores enteros. Por su parte, los intentos serán 6, y la bandera `hayGanador` inicia con un valor falso.

La función principal de nuestra clase, se llama **realizarIntento()**, en dónde se ingresará un valor y determinará si este es igual, menor o mayor al número secreto previamente generado. La función regresa un valor entero indicando si el número fue menor (-1), mayor (1) o igual (0) al valor secreto y permite, con esto, que la clase encargada de la interfaz gráfica de usuario pueda tomar las decisiones respecto a qué mostrar dependiendo el estado final de cada intento hecho por el usuario. La función se implementa de la siguiente manera:

```
public int realizarIntento(int valor) {  
    // Únicamente se permite que se hagan intentos si no ha ganado el usuario  
    // y si tiene más de 0 intentos disponibles.  
    if (!this.hayGanador && this.intentos > 0) {  
        if (valor == this.numero) {  
            this.hayGanador = true; // Colocamos la bandera en verdadero.  
        } else {  
            // Restamos un intento ya que no fue igual.  
            this.intentos--;  
  
            // Regresamos un valor dependiendo si es menor o mayor al número.  
            if (valor < this.numero) {  
                return -1; // El valor es menor.  
            } else {  
                return 1; // El valor es mayor.  
            }  
        }  
    }  
}
```

Cabe destacar que, si ya ha ganado o si el número de intentos es menor o igual a cero, ninguna acción será realizada. En caso de que aún no se haya ganado el juego y se tengan intentos disponibles, se procede a determinar en qué situación se encuentra el valor ingresado conforme a nuestro número secreto y retorna el valor apropiadamente.

La siguiente clase a analizar es **AdivinaNumeroGUI**, la cual contiene la implementación de la interfaz gráfica de usuario de nuestra aplicación. Nuestra interfaz gráfica de usuario es muy simple:



Los elementos dentro de la interfaz son:

- Ventana con el título "Adivina el Número".
- Etiqueta con los mensajes de "¡El número  $x$  es mayor!", "¡El número  $x$  es menor!" y "¡Has ganado con el número  $x$ !".
- Campo de texto para introducir el número a intentar.
- Botón para realizar la acción de comprobación.
- Etiqueta para mostrar el número de intentos disponibles.

Para generar cada uno de estos elementos en el orden presentado y asignarles un valor inicial, se utiliza el constructor de la clase, siendo:

```
public AdivinaNumeroGUI() {  
    // Creamos una instancia del juego.  
    this.juego = new AdivinaNumero();  
  
    // Opciones para la interfaz gráfica de usuario.  
    this.ventana = new JFrame("Adivina el Número"); // Creamos la ventana con el título "Adivina el Número"  
    this.ventana.getContentPane().setLayout(new FlowLayout());  
    // Establecemos FlowLayout para que todo esté alineado.  
  
    this.botonIntento = new JButton("Probar número");  
    this.botonIntento.addActionListener(this);  
  
    this.campoIntento = new JTextField("Número ... ",7);  
  
    this.etiquetaEstado = new JLabel("Realiza tu primer intento ...");  
  
    this.etiquetaNumeroIntento = new JLabel("Intentos disponibles: " + this.juego.obtenerIntentosDisponibles());  
  
    this.ventana.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);  
  
    // Acomodamos los elementos en el orden que deben aparecer en la ventana.  
    this.ventana.getContentPane().add(this.etiquetaEstado);  
    this.ventana.getContentPane().add(this.campoIntento);  
    this.ventana.getContentPane().add(this.botonIntento);  
    this.ventana.getContentPane().add(this.etiquetaNumeroIntento);  
  
    // Determinamos el tamaño de la ventana automáticamente a partir de los elementos.  
    this.ventana.pack();  
  
    // Se hace visible la ventana.  
    this.ventana.setVisible(true);  
}
```

Para la ejecución de nuestro programa, debemos *escuchar* los eventos que se generen cuando los usuarios hagan click en el botón “Probar Número”, por lo que subscribimos nuestra clase a los eventos generados por dicho botón e

implementamos la lógica que maneja los distintos estados de nuestro juego en función de cada evento que recibamos.

La implementación de la función que escucha dichos eventos es:

```
// Método de escucha de eventos a los que está suscrita la clase.
public void actionPerformed(ActionEvent evento) {
    // Revisamos si el usuario no ha adivinado el número y tiene intentos disponibles.
    if (!this.juego.hayGanador() && this.juego.obtenerIntentosDisponibles() > 0) {
        // Extraer el valor de la entrada.
        int valor = Integer.parseInt(this.campoIntento.getText());

        // Realizamos el intento con el valor de la entrada y almacenamos el valor de retorno.
        int resultadoIntento = this.juego.realizarIntento(valor);

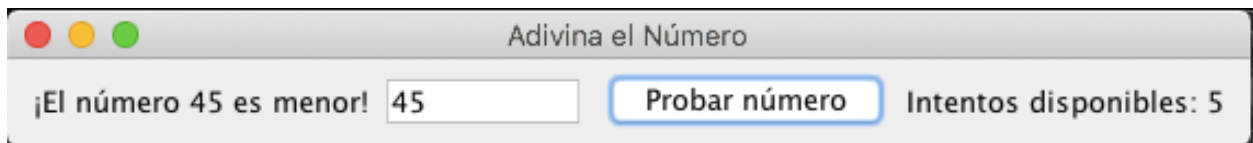
        // Si el valor de retorno es 0, el usuario ya ganó (porque verificamos previamente los intentos disponibles).
        if (resultadoIntento == 0) {
            this.etiquetaEstado.setText("¡Has adivinado el número " + valor + "!");
            this.etiquetaNumeroIntento.setText("Intentos disponibles: -");
        } else if (resultadoIntento == -1) {
            // Si el valor de retorno es -1, quiere decir que el usuario ingresó un número menor al secreto.
            this.etiquetaEstado.setText("¡El número " + valor + " es menor!");
            this.etiquetaNumeroIntento.setText("Intentos disponibles: " + this.juego.obtenerIntentosDisponibles());
        } else {
            // Si el valor de retorno es 1, quiere decir que el usuario ingresó un número mayor al secreto.
            this.etiquetaEstado.setText("¡El número " + valor + " es mayor!");
            this.etiquetaNumeroIntento.setText("Intentos disponibles: " + this.juego.obtenerIntentosDisponibles());
        }
    }

    // Verificamos si el usuario ya perdió para mostrar mensaje.
    if (this.juego.obtenerIntentosDisponibles() ≤ 0 && !this.juego.hayGanador()) {
        this.etiquetaEstado.setText("El número era: " + this.juego.obtenerNumeroSecreto() + ", lástima.");
    }
}
```

Y la lógica es muy simple. Si aún se cuentan con intentos disponibles y no se ha ganado anteriormente, se captura el valor que se encuentra en el campo de texto y procedemos a llamar a la función `realizarIntento()` de nuestra clase

**AdivinaNumero** con la que obtendremos el estado actual de nuestro juego, así

como una bandera indicativa de en qué estado se encuentra el valor ingresado (mayor, menor o igual al número secreto), en caso de que no se cuenten con turnos disponibles, mostraremos un mensaje de error y en caso de que se haya ganado mostraremos un texto indicando al usuario que ha ganado el juego. En caso de que el valor sea menor:

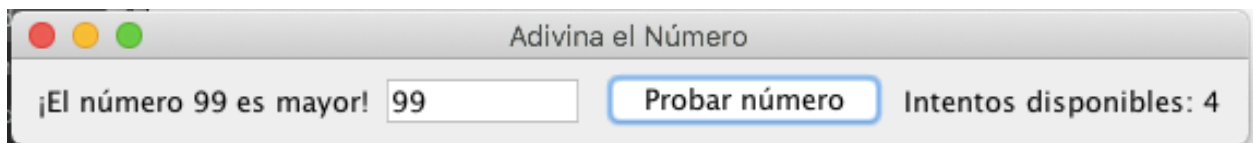


Adivina el Número

¡El número 45 es menor!

 Intentos disponibles: 5

En caso de que el valor sea mayor:

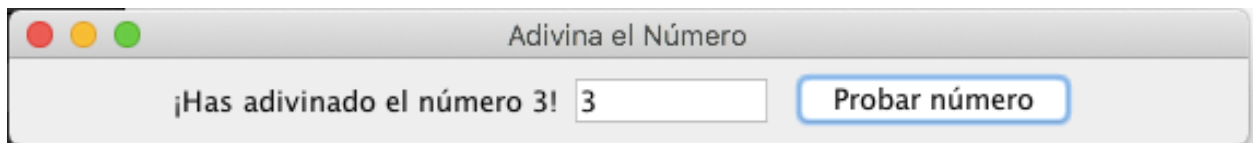


Adivina el Número

¡El número 99 es mayor!

 Intentos disponibles: 4

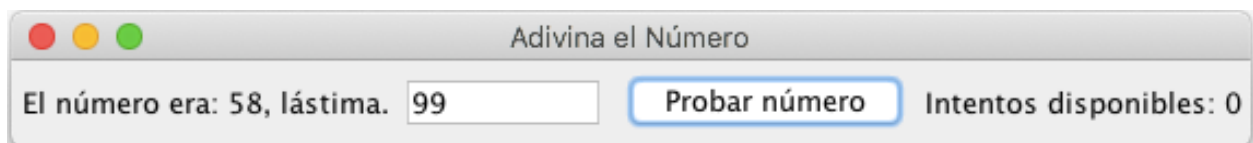
En caso de que ganemos:



Adivina el Número

¡Has adivinado el número 3!

En caso de que perdamos:



Adivina el Número

El número era: 58, lástima.

 Intentos disponibles: 0



## Conclusión

El desarrollo de este juego nos permite desarrollar una aplicación con interfaz gráfica que permite ofrecer un programa con mayor sencillez y utilidad para los usuarios finales ya que, a pesar de que es posible y factible desarrollar un programa cuya ejecución sea dentro de la terminal de nuestra computadora, las interfaces gráficas de usuario son elementos *cruciales* para desarrollar aplicaciones modernas ya que limita los factores de error a los cuales nuestro software está expuesto al ser utilizado por usuarios de distintos niveles de *conocimiento* en el uso de software y, en lo general, en el uso de computadoras. Para mí, es importante tratar de que el código de la interfaz y aquél con la lógica se encuentren separados. No solamente por las ventajas en un paradigma orientado a objetos como lo es el de Java, sino porque posteriormente se pueden desarrollar y analizar, de manera particular, las peculiaridades de cada módulo con el detalle que requiere cada uno.

Si bien el desarrollo de interfaces en sí mismo puede representar una labor para la cual se asignaría un equipo entero de programadores, es importante conocer la interacción que se tienen entre las distintas clases, así como los datos que requerirá la interfaz para poder ejecutarse de mejor manera.

Con esta práctica, se vuelve evidente la separación de cada uno de estos elementos así como el entendimiento general del por y para qué hemos implementado cada uno de los métodos y variables empleados a lo largo de nuestro programa.