Future Developer Week

Javier San Juan Cervera

Contenido

Día 4: Implementando la funcionalidad

- Condicionales
- Estructuras de datos
- Arrays

Condicionales

Condicionales

- Hasta ahora, hemos aprendido a escribir una lista de sentencias que se ejecutan una detrás de otra.
- Pero cuando programamos, es a veces necesario ejecutar una serie de sentencias sólo si se cumple una determinada condición.
- Por ejemplo: escribimos un programa que realiza la división entre dos números pedidos al usuario.

Condicionales

```
Console.Write("Escriba el numerador: ");
int numerador = int.Parse(Console.ReadLine());

Console.Write("Escriba el denominador: ");
int denominador = int.Parse(Console.ReadLine());

Console.WriteLine("Resultado: " + numerador / denominador);

Console.ReadKey();
```

NOTA: La función int.Parse convierte un string en un int.

Condicionales

- Si escribimos el código anterior en la función `static void Main(string[] args)` de una aplicación de consola, ejecutamos, e introducimos **0** como **denominador**, se produce un error de ejecución.
- Dicho error es **“Intento de dividir por cero”**, y el compilador nos aconseja asegurarnos de que el denominador de una fracción no sea cero antes de realizar la división.
- Para poder comprobar el valor de una expresión y actuar en consecuencia, utilizamos los **condicionales**.

Condicionales

```
Console.Write("Escriba el numerador: ");
int numerador = int.Parse(Console.ReadLine());

Console.Write("Escriba el denominador: ");
int denominador = int.Parse(Console.ReadLine());

if (denominador != 0)
{
      Console.WriteLine("Resultado: " + numerador / denominador);
}

Console.ReadKey();
```

Condicionales

- Si ejecutamos ahora la aplicación, veremos que ahora sólo mostrará el resultado de la división si el segundo valor introducido es distinto de cero.
- Esto es porque hemos encerrado la sentencia `Console.WriteLine("Resultado: " + numerador / denominador);` dentro del bloque `if (denominador != 0)`.
- El operador `!=` nos dice si dos expresiones son diferentes. También se puede comprobar:
  - `==` Si dos expresiones son iguales.
  - `<` Si la de la izquierda es menor que la de la derecha.
  - `>` Si la de la izquierda es mayor que la de la derecha.
  - `<=` Si la de la izquierda es menor o igual que la de la derecha.
  - `>=` Si la de la izquierda es mayor o igual que la de la derecha.

Condicionales

- El bloque `if` permite **ejecutar** las sentencias que contiene únicamente si la expresión entre paréntesis **es cierta**.
- Si sólo debe ejecutarse **una sentencia**, se pueden **omitir las llaves**, pero siempre son necesarias si se deben ejecutar varias sentencias (se pondrán todas entre las llaves).

Booleanos

- Hemos visto distintos tipos de expresiones hasta ahora. Algunas devolvían un valor entero, otras una valor decimal, otras un valor de tipo string…
- Existen expresiones que como valor devuelven **cierto** (`true`) o **falso** (`false`). Son las llamadas **expresiones booleanas**.
- Este tipo es precisamente el que devuelven los operadores `==`, `!=`, `<`, `>`, `<=`, `>=`.
- El tipo de estas expresiones es `bool`, y se puede utilizar como tipo en variables y funciones.

Else

- Volviendo al ejemplo de la división entre cero, recordemos que la división sólo se realizaba si el denominador era distinto de cero.
- En el caso de que la **condición no se cumpla**, podríamos indicarle al usuario esta circunstancia.
- Una forma de hacerlo podría ser la siguiente:

ELSE

```
Console.Write("Escriba el numerador: ");
int numerador = int.Parse(Console.ReadLine());

Console.Write("Escriba el denominador: ");
int denominador = int.Parse(Console.ReadLine());

if (denominador != 0)
{
     Console.WriteLine("Resultado: " + numerador / denominador);
}
if (denominador == 0)
{
     Console.WriteLine("No es posible realizar una división entre cero");
}

Console.ReadKey();
```

Else

- Pero existe una forma mejor, que no requiere comprobar una segunda condición.
- Decíamos que si la condición de un `if` **es cierta**, entonces se ejecuta el código entre las llaves a continuación.
- También es posible indicarle a un `if` las sentencias a ejecutar **cuando la condición no es cierta** utilizando `else`:

Else

```
Console.Write("Escriba el numerador: ");
int numerador = int.Parse(Console.ReadLine());

Console.Write("Escriba el denominador: ");
int denominador = int.Parse(Console.ReadLine());

if (denominador != 0)
{
     Console.WriteLine("Resultado: " + numerador / denominador);
}
else
{
     Console.WriteLine("No es posible realizar una división entre cero");
}

Console.ReadKey();
```

Else if

- Por último, es también posible encadenar la comprobación de varias condiciones, y ejecutar únicamente las sentencias de la condición que se cumpla.
- Esto lo hacemos con `else if`, que se coloca tras las sentencias de un `if`, y nos permite comprobar una nueva condición si la anterior no se cumplió:

ELSE IF

```
Console.Write("Introduzca el primer valor: ");
int a = int.Parse(Console.ReadLine());

Console.Write("Introduzca el segundo valor: ");
int b = int.Parse(Console.ReadLine());

if (a == b)
{
      Console.WriteLine("a es igual a b");
}
else if (a < b)
{
      Console.WriteLine("a es menor a b");
}
else if (a > b)
{
      Console.WriteLine("a es mayor a b");
}

Console.ReadKey();
```

Comprobando si la respuesta es correcta en U-Quiz

Comprobando si la respuesta es correcta en U-Quiz

- Ahora que sabemos comprobar condiciones, vamos a informar al usuario de si la respuesta que ha seleccionado es correcta.
- Si el usuario pulsa el botón de la aplicación, debemos comprobar si el RadioButton que está seleccionado se corresponde con el de la respuesta correcta.
- Para saber si un RadioButton está seleccionado, podemos consultar el valor de su propiedad `Checked`.

Estructuras de datos

Estructuras de datos

Si quisiéramos almacenar en variables los datos de una pregunta de U-Quiz, es posible que hiciésemos algo como esto:

```
string Texto = “¿Cuál es la capital de California?”;
string Opcion1 = “Los Ángeles”;
string Opcion2 = “San Francisco”;
string Opcion3 = “Sacramento”;
string Opcion4 = “San Diego”;
Image Imagen = Properties.Resources.california;
int OpcionCorrecta = 3;
```

Estructuras de datos

- Ya hemos visto algunos tipos de datos, como `int`, `double`, `string` y `bool`. Estos son llamados **tipos primitivos**, y almacenan **un valor**.
- En cambio, hemos trabajado también con otros tipos. Por ejemplo, los botones de nuestro formulario son del tipo `Button`; las etiquetas, del tipo `Label`; las cajas de texto, de tipo `TextBox`, etc.
- Hemos visto que estos tipos tienen distintas **propiedades**, cada una con su valor, como `Text`, `Visible` o `Image`.
- Estos son los llamados **tipos compuestos** o **estructuras de datos**.
- Podemos **crear** los **nuestros** propios.

Estructuras de datos

Así, podríamos crear una estructura para guardar los datos de nuestra pregunta:

```
struct Pregunta
{
    public string Texto;
    public string Opcion1;
    public string Opcion2;
    public string Opcion3;
    public string Opcion4;
    public Image Imagen;
    public int OpcionCorrecta;
}
```

Estructuras de datos

Y tener todos los datos de la pregunta en una sola variable con varias propiedades:

```
Pregunta pregunta1;
pregunta1.Texto = “¿Cuál es la capital de California?”;
pregunta1.Opcion1 = “Los Ángeles”;
pregunta1.Opcion2 = “San Francisco”;
pregunta1.Opcion3 = “Sacramento”;
pregunta1.Opcion4 = “San Diego”;
pregunta1.Imagen = Properties.Resources.california;
pregunta1.OpcionCorrecta = 3;
```

Creando una estructura de datos en U-Quiz

Creando una estructura de datos en U-Quiz

- Vamos a **crear la estructura** `Pregunta` en U-Quiz.
- En el evento `Form1_Load` que se genera al hacer doble click sobre el formulario, vamos a crear una variable del tipo `Pregunta` y a darle valor a sus propiedades.
- Después, vamos a **definir una función** `MostrarPregunta` que reciba como parámetro una pregunta y cambie las etiquetas, radio buttons e imágenes para que utilice los datos de la pregunta. Llamaremos a esta función desde `Form1_Load`.

Arrays

Arrays

- Nuestro juego U-Quiz debe tener varias preguntas.
- Con lo visto hasta, podríamos tener varias variables `pregunta1`, `pregunta2`, etc, de tipo `Pregunta`, y una variable entera `preguntaActual` que indica qué pregunta se muestra actualmente.
- Cuando pulsamos el botón de la aplicación, nos dice si hemos acertado o no, incrementa la variable `preguntaActual`, y muestra la nueva pregunta en la interfaz:

Arrays

```
preguntaActual = preguntaActual + 1;
if (preguntaActual == 1)
{
      MostrarPregunta(pregunta1);
}
else if (preguntaActual == 2)
{
      MostrarPregunta(pregunta2);
}
else if (preguntaActual == 3)
{
      MostrarPregunta(pregunta3);
}
…
```

Arrays

- Existe una forma de almacenar todas las preguntas en una única variable, y es utilizando **colecciones**.
- Existen varios tipos de colecciones, como los arrays, las listas, los mapas…
- En este curso, vamos a aprender a utilizar los **arrays**.
- Un array es, simplemente, una colección de datos a la que podemos acceder a través de una variable y un índice.

Arrays

- Para declarar una variable que pueda contener un array, hacemos lo siguiente:

```
int[] unArray;
```

- Este ejemplo crea una variable que permite almacenar un array de enteros. Para guardar un nuevo array en la variable, hacemos:

```
unArray = new int[10];
```

- Lo que guardará en la variable un nuevo array de 10 enteros.

Arrays

- Para dar valor a un de los elementos del array, hacemos:

```
miArray[0] = 25;
```

- Lo que da el valor 25 al primer elemento del array.
- Al valor entre corchetes se le llama **índice**, y comienza en **cero**.
- Para obtener el valor de un elemento del array, hacemos:

```
Console.WriteLine(miArray[5]);
```

- Que imprime el valor del sexto elemento del array.
- Podemos obtener el **número de elementos** de un array con la propiedad Length:

```
Console.WriteLine(miArray.Length);
```

Definiendo varias preguntas en U-Quiz

Definiendo varias preguntas en U-Quiz

- Ahora que sabemos manejar arrays, se simplifica la gestión de varias preguntas.
- Utilizaremos un array para guardar todas las preguntas, y una variable entera para guardar el índice de la pregunta actual.
- Cada vez que respondemos, se incrementa el índice:
  - Si no hemos llegado al final del array de preguntas, mostramos la siguiente.
  - En caso contrario, mostramos el número de preguntas acertadas (con lo que necesitaremos otra variable que se vaya incrementando por cada pregunta que acertemos).