👀

EXCEPCIONES

> La lógica de un programa orientado a objetos se implementa mediante la creación de objetos, el envío de mensajes a esos objetos y el uso de estructuras de control de flujo para gestionar esos mensajes. Al enviar un mensaje, esperamos que el estado del programa cambie de acuerdo con el comportamiento del método que se ejecuta; si esto no sucede, se considera que el programa es incorrecto.

Se definen varios términos para aclarar conceptos relacionados con errores en un programa:

- **Equivocación (mistake)**: una acción humana que produce un resultado incorrecto.
- **Falta (fault)**: una definición incorrecta en un paso, proceso o dato, que se origina de una equivocación.
- **Falla (failure)**: un comportamiento o resultado incorrecto, que resulta de una falta.
- **Error**: la diferencia entre un resultado calculado y el valor verdadero o especificado.

Una equivocación puede llevar a un error. Por ejemplo, si un usuario ingresa incorrectamente la ubicación de un archivo, el programa no podrá abrirlo, pero al solicitarle que intente nuevamente, podría ingresar la ubicación correcta. Esto se maneja como una excepción (por ejemplo, **FileNotFoundException**) que se puede gestionar para permitir al usuario corregir su entrada.

Sin embargo, si el error proviene de un programador (por ejemplo, un objeto nulo al que se intenta enviar un mensaje), esto resulta en una falla (por ejemplo, **NullReferenceException**), lo que significa que el programa no puede continuar sin una corrección en el código.

La decisión de si una excepción es una falla depende del contexto y de cómo el programa maneja la situación. Para gestionar excepciones en C#, se utiliza una estructura de control **try...catch...finally**. El bloque **try** contiene el código donde puede ocurrir una excepción, el bloque **catch** maneja la excepción si ocurre, y el bloque **finally** asegura que ciertos códigos se ejecuten siempre, como liberar recursos.

Por ejemplo, al abrir un archivo para escritura, el código que maneja la apertura y escritura del archivo se coloca en el bloque **try**. Si ocurre una excepción de acceso no autorizado (**UnauthorizedAccessException**), se captura en el bloque **catch**. Finalmente, el bloque **finally** se utiliza para cerrar el archivo, independientemente de si ocurrió una excepción o no, asegurando que se liberen los recursos.

```
FileStream file = null;
FileInfo fileinfo = new FileInfo(@"c:\file.txt");
```

```
try
{
file = fileinfo.OpenWrite();
file.WriteByte(0xF);
}
catch (UnauthorizedAccessException e)
{
Debug.WriteLine("Acceso no autorizado");
throw e;
}
```

```
finally
{
if (file != null)
{
file.Close();
}
}
```

- En C#, todas las excepciones heredan de la clase base **System.Exception**. Para lanzar una excepción, se utiliza la palabra clave **throw** seguida de la clase de excepción correspondiente. En el bloque **catch**, se puede usar **throw** sin argumentos para relanzar la excepción que se está capturando.
- El bloque **catch** puede incluir o no una clase de excepción. Si no incluye una clase, cualquier excepción en el bloque **try** activa el bloque **catch**. Si se especifica una clase, solo las excepciones de esa clase en el bloque **try** provocan la ejecución del bloque **catch**, permitiendo recibir un objeto de la clase de excepción, que contiene información adicional como un mensaje. Es posible tener múltiples bloques **catch** para manejar excepciones de diferentes clases. Si se produce una excepción sin un bloque **try...catch...finally**, el programa busca un bloque en el método que invocó el método donde ocurrió la excepción, continuando hasta llegar al programa principal. Si no se maneja la excepción, el programa termina.

Por ejemplo, en un programa que gestiona una biblioteca representada por la clase **Library**, que contiene obras musicales (canciones y películas), se puede evitar que se agreguen canciones de un género no deseado, como la marcha. Para ello, se puede lanzar una excepción cuando se intenta agregar una canción de ese género. Para implementar esto, primero se define una nueva clase de excepción llamada **LibraryException**. Aunque no es necesario crear nuevas clases de excepción si las existentes son adecuadas, se hace en este caso para ilustrar el proceso.

```
[Serializable]
internal class LibraryException : Exception
{
public LibraryException()
```

```
{
}
```

```
public LibraryException(string message)
: base(message)
{
}

public LibraryException(string message, Exception innerExce
ption)
: base(message, innerException)
{
}

protected LibraryException(SerializationInfo info, Streamin
gContext context)
: base(info, context)
{
}
```

Para ser sucesora de `Exception` nuestra `LibraryException` debe implementar constructores sin parámetros, con el mensaje como parámetro, con un mensaje y una excepción encadenada, y debe ser decorada con el atributo `[Serializable]`.

Como el objetivo es evitar crear instancias de `Song` con género "marcha", el constructor tira una excepción si corresponde. Vean el código a continuación, especialmente la sentencia `throw`. Recuerden que la excepción es un objeto, por es necesario usar `new` para crearla.

```
public class Song : Media
{
private string artist;
```

```
public Song(string name, string genre, string artist)
: base(name, genre)
{
    if (string.Equals(genre, "marcha", StringComparison.Ord
inalIgnoreCase))
    {
        throw new LibraryException($"No me gusta {name} por
que no me gusta la marcha");
    }
    this.artist = artist;
}
```

- El programa principal agrega varias canciones, protegemos la posibilidad de ocurrencia de una excepción en un bloque `try`; si ocurre una excepción, en el bloque `catch` simplemente se imprime por consola el mensaje de la excepción; en cualquier caso, en el bloque `finally`, se imprime al final el contenido de la biblioteca:

```
public class Program
{
private static Library library = new Library();
```

```
public static void Main(string[] args)
{
    try
    {
        AddMovies();
        AddSongs();
        AddMarchaSongs();
    }
    catch (LibraryException e)
    {
        Console.WriteLine(e.Message);
    }
    finally
    {
        PrintMediaItems("La biblioteca contiene:", librar
```

```
y);
    }
}
```