Método `Registrar(Producto obj, out string Mensaje)` – Explicación paso a paso

Este documento explica en detalle qué hace el método `Registrar` en C#, que se utiliza para **registrar (insertar) un nuevo producto** en la base de datos usando un **procedimiento almacenado**.

---

1. Código completo del método

```
// Parametros de entrada y salida - "obj" objeto declarado de tipo Producto
public int Registrar(Producto obj, out string Mensaje)
{
    int idproductogenerado = 0;
    Mensaje = string.Empty;

    try
    {
        // Realiza la conexion a la base de datos con la cadena de conexion
        using (SqlConnection oconexion = new SqlConnection(Conexion.cadena))
        {
            // Recibe como parametro el nombre del procedimiento almacenado
            SqlCommand cmd = new
SqlCommand("SP_REGISTRARPRODUCTO".ToString(), oconexion);

            // Parametros de entrada
            cmd.Parameters.AddWithValue("Codigo", obj.Codigo);
            cmd.Parameters.AddWithValue("Nombre", obj.Nombre);
            cmd.Parameters.AddWithValue("Descripcion", obj.Descripcion);
            cmd.Parameters.AddWithValue("IdCategoria",
obj.oCategoria.IdCategoria);

            cmd.Parameters.AddWithValue("Stock", obj.Stock);
            cmd.Parameters.AddWithValue("Precio", obj.Precio);

            cmd.Parameters.AddWithValue("Estado", obj.Estado);


            // Parametros de salida
            cmd.Parameters.Add("Resultado", SqlDbType.Int).Direction =
ParameterDirection.Output;
            cmd.Parameters.Add("Mensaje", SqlDbType.VarChar, 500).Direction
= ParameterDirection.Output;

            cmd.CommandType = CommandType.StoredProcedure;
            oconexion.Open();

            cmd.ExecuteNonQuery();
```

```
            idproductogenerado =
Convert.ToInt32(cmd.Parameters["Resultado"].Value);
            Mensaje = cmd.Parameters["Mensaje"].Value.ToString();
        }
    }
    catch (Exception ex)
    {
        idproductogenerado = 0;
        Mensaje = ex.Message;
    }

    return idproductogenerado;
}
```

---

2. Idea general del método

El método `Registrar` se usa para **insertar un nuevo producto en la base de datos** usando un **procedimiento almacenado** llamado `SP_REGISTRARPRODUCTO`.

Hace lo siguiente:

1. Recibe un objeto `Producto` (`obj`) con los datos cargados.
2. Llama al procedimiento almacenado `SP_REGISTRARPRODUCTO` en SQL Server.
3. Le pasa los valores del producto como **parámetros de entrada**.
4. Recibe desde el procedimiento:
5. Un **id generado** para el nuevo producto (`Resultado`).
6. Un **mensaje** (`Mensaje`) indicando si todo salió bien o si hubo algún problema.
7. Devuelve el **id del producto generado** y llena el parámetro `out Mensaje` con el mensaje correspondiente.

En resumen: es un método para **registrar un producto nuevo** y saber si se guardó correctamente.

---

3. Parámetros del método y variables iniciales

```
public int Registrar(Producto obj, out string Mensaje)
{
    int idproductogenerado = 0;
    Mensaje = string.Empty;
    ...
}
```

Parámetros del método

- `Producto obj`: objeto que contiene la información del producto a registrar.

- `out string Mensaje`: parámetro de salida. Al terminar el método, contendrá un mensaje que viene desde el procedimiento almacenado (por ejemplo: "Registro exitoso" o un mensaje de error controlado).

Variables iniciales

- `idproductogenerado = 0;`
- Variable que guardará el **ID** que devuelve el procedimiento almacenado.
- Si algo falla, queda en `0`.
- `Mensaje = string.Empty;`
- Inicialmente se define el mensaje como vacío.

---

4. Conexión a la base de datos y `try/catch`

```
try
{
    using (SqlConnection oconexion = new SqlConnection(Conexion.cadena))
    {
        ...
    }
}
catch (Exception ex)
{
    idproductogenerado = 0;
    Mensaje = ex.Message;
}
```

- El `try` contiene todo el código que accede a la base de datos.
- `using (SqlConnection ...)` se encarga de abrir y cerrar la conexión correctamente.
- Si ocurre alguna excepción (error de servidor, desconexión, etc.), se captura en el `catch`:
- `idproductogenerado` se pone en `0`.
- `Mensaje` se llena con `ex.Message` (el detalle del error técnico).

---

5. Creación del `SqlCommand` con el procedimiento almacenado

```
SqlCommand cmd = new SqlCommand("SP_REGISTRARPRODUCTO".ToString(),
oconexion);
```

- Se crea un objeto `SqlCommand` llamado `cmd`.
- El primer parámetro es el **nombre del procedimiento almacenado**: `SP_REGISTRARPRODUCTO`.
- El segundo parámetro es la conexión `oconexion`.

Nota: el `.ToString()` sobre el literal `"SP_REGISTRARPRODUCTO"` no es necesario, ya que ya es un string. Podría ser simplemente:

```
SqlCommand cmd = new SqlCommand("SP_REGISTRARPRODUCTO",
oconexion);
```

Más abajo se indica que este comando es de tipo `StoredProcedure`:

```
cmd.CommandType = CommandType.StoredProcedure;
```

---

6. Parámetros de entrada

```
cmd.Parameters.AddWithValue("Codigo", obj.Codigo);
cmd.Parameters.AddWithValue("Nombre", obj.Nombre);
cmd.Parameters.AddWithValue("Descripcion", obj.Descripcion);
cmd.Parameters.AddWithValue("IdCategoria", obj.oCategoria.IdCategoria);

cmd.Parameters.AddWithValue("Stock", obj.Stock);
cmd.Parameters.AddWithValue("Precio", obj.Precio);

cmd.Parameters.AddWithValue("Estado", obj.Estado);
```

Estos son los **parámetros que se envían al procedimiento almacenado**.

- `"Codigo"` → valor tomado de `obj.Codigo`.
- `"Nombre"` → valor tomado de `obj.Nombre`.
- `"Descripcion"` → `obj.Descripcion`.
- `"IdCategoria"` → `obj.oCategoria.IdCategoria`.
- `"Stock"` → `obj.Stock`.
- `"Precio"` → `obj.Precio`.
- `"Estado"` → `obj.Estado`.

Es importante que estos nombres coincidan con los parámetros que espera el procedimiento `SP_REGISTRARPRODUCTO` en SQL Server.

Ejemplo de cómo podría verse el procedimiento almacenado:

```
CREATE PROCEDURE SP_REGISTRARPRODUCTO
    @Codigo      VARCHAR(50),
    @Nombre      VARCHAR(100),
    @Descripcion VARCHAR(500),
    @IdCategoria INT,
    @Stock       INT,
    @Precio      DECIMAL(18,2),
```

```
    @Estado      BIT,
    @Resultado   INT OUTPUT,
    @Mensaje     VARCHAR(500) OUTPUT
AS
BEGIN
    -- Lógica para insertar el producto y setear @Resultado y @Mensaje
END;
```

(El código exacto puede variar, pero la idea es esa.)

---

7. Parámetros de salida

```
cmd.Parameters.Add("Resultado", SqlDbType.Int).Direction =
ParameterDirection.Output;
cmd.Parameters.Add("Mensaje", SqlDbType.VarChar, 500).Direction =
ParameterDirection.Output;
```

Acá se declaran **parámetros de salida** (`OUTPUT`) que el procedimiento almacenado va a completar:

- `"Resultado"`:
- Tipo `Int`.
- Dirección `Output`.

- Normalmente se usa para devolver el **ID del producto generado** o un código de resultado (por ejemplo, 0 = error, >0 = OK).

- `"Mensaje"`:

- Tipo `VarChar` de longitud 500.
- Dirección `Output`.
- Se usa para enviar un **mensaje de texto** desde el procedimiento (éxito, error controlado, etc.).

En el procedimiento almacenado, estos parámetros se declaran como `OUTPUT` y se asignan dentro del código SQL.

---

8. Tipo de comando y apertura de la conexión

```
cmd.CommandType = CommandType.StoredProcedure;
oconexion.Open();
```

- `CommandType.StoredProcedure` indica que `cmd.CommandText` es el **nombre de un procedimiento almacenado**, no una consulta SQL directa.
- `oconexion.Open()` abre la conexión con la base de datos.

---

9. Ejecución del comando

```
cmd.ExecuteNonQuery();
```

- Como se trata de un procedimiento almacenado que **no devuelve filas** (no es un `SELECT` que retorne un resultado tabular), se usa `ExecuteNonQuery()`.
- Este método ejecuta el procedimiento y:
- Realiza la operación de inserción.
- Llena los parámetros de salida (`Resultado`, `Mensaje`).

---

10. Lectura de los parámetros de salida

```
idproductogenerado = Convert.ToInt32(cmd.Parameters["Resultado"].Value);
Mensaje = cmd.Parameters["Mensaje"].Value.ToString();
```

Después de ejecutar `ExecuteNonQuery()`, los parámetros de salida ya tienen valores.

- `cmd.Parameters["Resultado"].Value` contiene el valor que el procedimiento almacenado asignó a `@Resultado`.
- Se convierte a `int` y se guarda en `idproductogenerado`.
- `cmd.Parameters["Mensaje"].Value` contiene el texto que el procedimiento asignó a `@Mensaje`.
- Se convierte a `string` y se devuelve al exterior mediante el parámetro `out Mensaje`.

De esta forma, el método puede informar **qué pasó** durante el registro.

---

11. Manejo de errores en el `catch`

```
catch (Exception ex)
{
    idproductogenerado = 0;
    Mensaje = ex.Message;
}
```

Si se produce cualquier excepción:

- `idproductogenerado` queda en `0` → señal de que no se pudo registrar correctamente.
- `Mensaje` se llena con el mensaje de error técnico (`ex.Message`).

En una aplicación final, muchas veces este mensaje se muestra al usuario o se registra en logs.

---

12. Valor de retorno del método

```
return idproductogenerado;
```

El método devuelve el valor de `idproductogenerado`:

- Si todo salió bien y el procedimiento almacenado lo hizo correctamente:
- `idproductogenerado` será el **ID del producto recién insertado**.
- Si algo falló:
- `idproductogenerado` será `0`.
- `Mensaje` contendrá información del error.

---

13. Resumen final

- **Propósito del método**: registrar un nuevo producto en la base de datos mediante el procedimiento almacenado `SP_REGISTRARPRODUCTO`.
- **Entrada**: un objeto `Producto` con todos los datos llenos.
- **Salida**:
- `int` → ID del producto generado (o 0 si falló).
- `out string Mensaje` → mensaje descriptivo del resultado.
- **Pasos internos**:
- Crear conexión a la base con `SqlConnection`.
- Crear un `SqlCommand` apuntando al procedimiento almacenado.
- Cargar los parámetros de entrada desde `obj`.
- Declarar y preparar parámetros de salida.
- Abrir la conexión.
- Ejecutar con `ExecuteNonQuery()`.
- Leer los parámetros de salida y asignarlos a `idproductogenerado` y `Mensaje`.
- Manejar posibles excepciones con `try/catch`.

Este patrón es muy común cuando trabajás con **procedimientos almacenados** en SQL Server desde C#, usando ADO.NET.