Capítulo 12
Gestión de excepciones y seguimiento

Objetivo

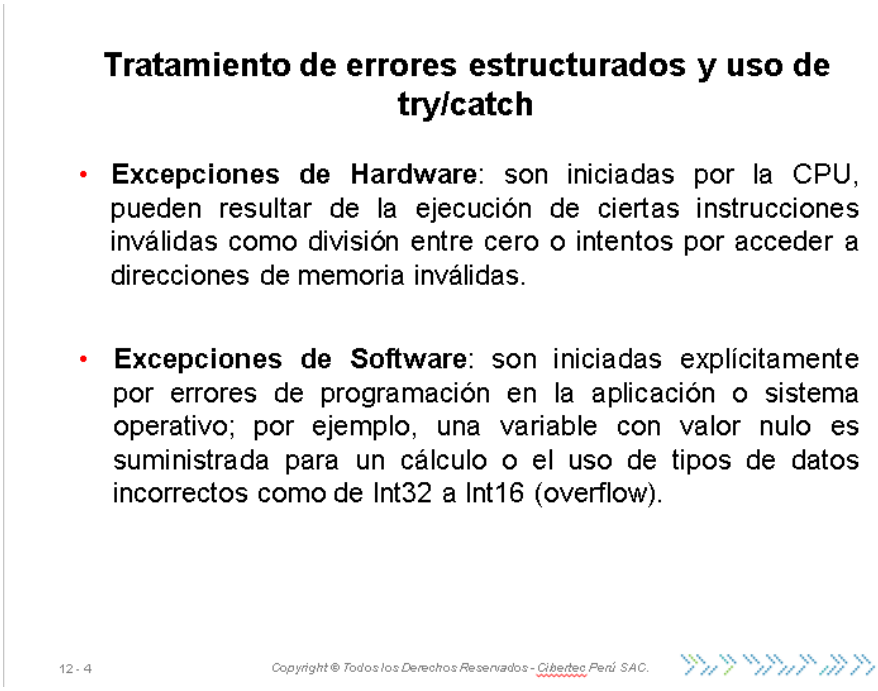
Al finalizar el capítulo, el alumno:

- Implementar el manejo de error en las aplicaciones Web.
- Usar las trazas durante pruebas y depuración de la aplicación.

Temas

1. Tratamiento de errores estructurados y uso de Try/Catch
2. Eventos Page_Error y Application_Error
3. Configuraciones en el archivo Web.config
4. Seguimiento de la aplicación Web

1. Tratamiento de errores estructurados y uso de Try/Catch



Una excepción se define como un evento que altera el flujo normal de ejecución de una aplicación, y, por lo tanto, requiere su identificación y tratamiento especial para cada caso. Las excepciones se agrupan en dos categorías principales:

- **Excepciones de Hardware:** son iniciadas por la CPU, pueden resultar de la ejecución de ciertas instrucciones inválidas como división entre cero o intentos por acceder a direcciones de memoria inválidas.

- **Excepciones de Software:** son iniciadas explícitamente por errores de programación en la aplicación o sistema operativo; por ejemplo, una variable con valor nulo es suministrada para un cálculo o el uso de tipos de datos incorrectos como de **Int32** a **Int16** (overflow).

El manejo de errores estructurados es un mecanismo que utilizan los lenguajes orientados a objetos para encapsular un bloque de código sensible a error, dentro del flujo normal de ejecución del programa. De esta manera, el programador podrá tener un control completo de los errores que se pudieran presentar. Al igual que muchos lenguajes de programación como C++, Java, Delphi, etc., en .Net se utilizan los bloques **Try..Catch..Finally**.

```
public ProductCategory GetProductCategoryWithExceptionHandling(int? id)
{
   var awContext = new AwContext();
   try
   {
  //Generando una excepción a propósito
      throw new Exception("Excepcion general");
      return new ProductCategory(awContext.ProductCategories.Find(id));
   }
   catch (InvalidOperationException ex)
   {
      throw;
   }
   catch (ArgumentNullException ex)
   {
      throw;
   }
   catch (Exception ex)
   {
      throw;
   }
   finally
   {
      awContext.Dispose();
   }
}
```

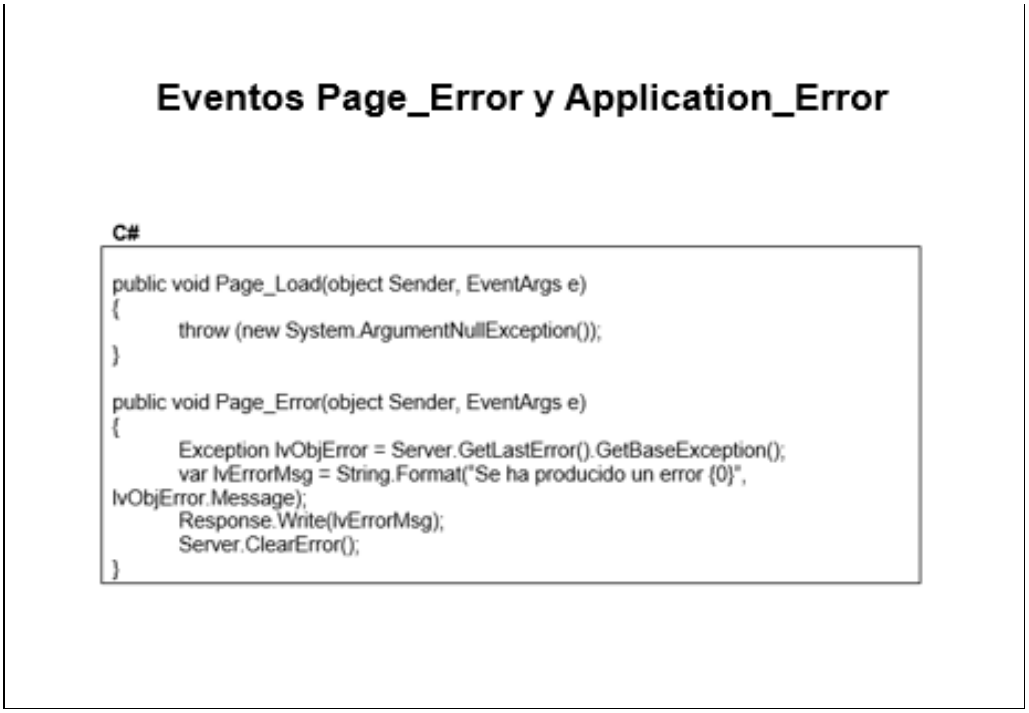
- **Try** (Obligatorio): indica al compilador el inicio de un bloque estructurado de excepción, dentro de esta sección se incluye el fragmento de código sensible considerando su comportamiento normal o esperado.

- **Catch** (Opcional, si hay Finally): indica al compilador que se ha desencadenado un error, y por lo tanto, solicita que se utilice el bloque de código de esta sección en lugar del error desencadenado. Pueden existir un sinnúmero de instrucciones catch, en donde cada una, está dirigida a controlar un tipo de excepción en particular; por ejemplo, **IOException** se usa para controlar problemas de acceso al disco por parte del manejo de archivos, mientras que **NullReferenceException** se utiliza para controlar el uso de valores nulos en el cálculo.

  No siempre es posible anticipar cada excepción potencial en tiempo de ejecución. Por ello, una técnica comúnmente usada, es incluir un bloque Catch que capture el objeto genérico Exception, luego de los bloques Catch para excepciones específicas. Este bloque Catch manipula cualquier excepción no contemplada inicialmente.

- **Finally** (Opcional si hay Catch): indica al compilador que siempre ejecute el bloque de código de esta sección, haya o no error.

2. Eventos Page_Error y Application_Error



Manejador de evento Page_Error

La clase **Page** provee el evento **Page_Error** para manipular todas las excepciones no controladas en una página. Si se necesita mostrar información al usuario acerca de la excepción, se debe usar el método **Response.Write**. No se puede mostrar información acerca del error en los controles Web porque el evento **Page_Error** evita que los controles se dibujen en la página.

Se puede acceder a los detalles del error en el evento Page_Error, invocando al método **GetLastError** del objeto Server. El siguiente fragmento de código muestra cómo implementar el método Page_Error:

C#

```
public void Page_Load(object Sender, EventArgs e)
{
  //Excepción generada intencionalmente para probar el manejo de excepciones.
  throw (new System.ArgumentNullException());
}

public void Page_Error(object Sender, EventArgs e)
{
  Exception lvObjError = Server.GetLastError().GetBaseException();
  var lvErrorMsg = String.Format("Se ha producido un error {0}", lvObjError.Message);
  Response.Write(lvErrorMsg);
  Server.ClearError();
}
```

VB.NET

```
Sub Page_Load(Sender as object, e as EventArgs)
        'Excepción generada intencionalmente para probar el manejo de excepciones.
         Throw(New System.ArgumentNullException())
End Sub

Sub Page_Error(Sender as object, e as EventArgs)
        Dim lvObjError as Exception = Server.GetLastError().GetBaseException()
        Dim lvErrorMsg As String = & _
              String.Format("<b>Se ha producido el siguiente error: </b> '{0}' ",  & _
                        lvObjError.Message)
        Response.Write(lvErrorMsg)
        Server.ClearError()
End Sub
```

Manejo de errores a nivel de aplicación

Otra técnica para atrapar excepciones no manejadas, es definir manejadores de evento de errores a nivel de aplicación. Existen dos opciones que se pueden seguir cuando se implementa un manejador de error a nivel de aplicación:

- Crear un método **Application_Error** en el archivo **Global.asax** de la aplicación Web. Se puede escribir código para el tratamiento de todas las excepciones no manejadas en este método de evento.

El siguiente fragmento de código muestra cómo implementar el método **Application_Error** en el archivo **Global.asax**:

C#

```
public void Application_Error(object sender, EventArgs e)
{
   // Código que se ejecuta ante errores no manejados.
}
```

VB.NET

```
<script runat="server">

   Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
      ' Código que se ejecuta ante errores no manejados.
   End Sub

</script>
```

3. Configuración en el archivo Web.config

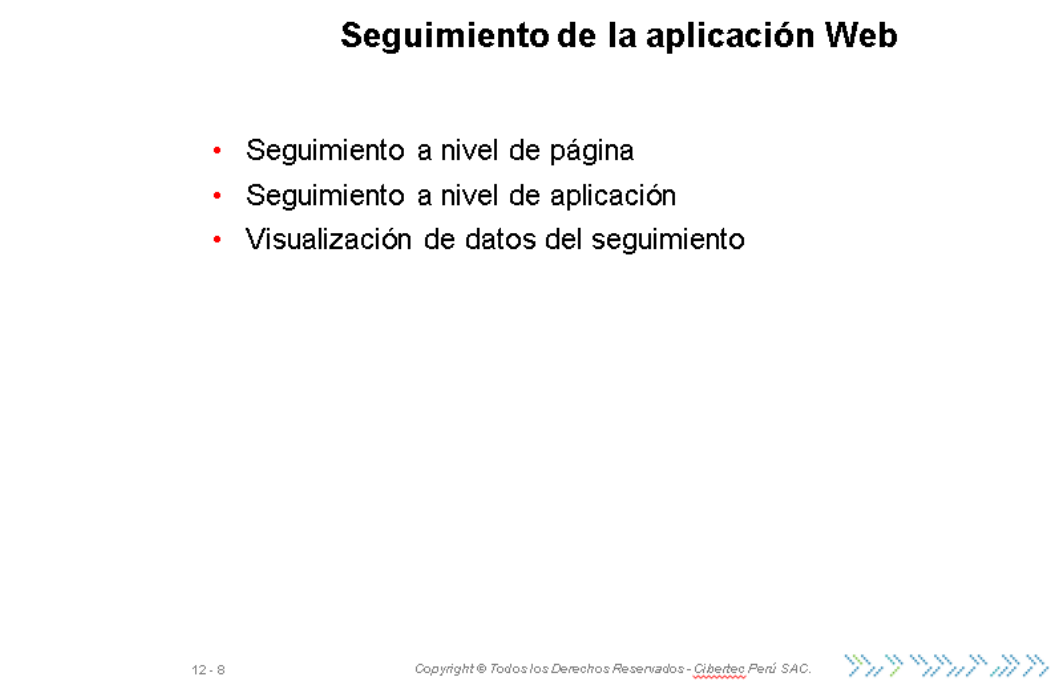
Para excepciones HTTP como “Página no encontrada” (Error 404), “Acceso no autorizado” (Error 401), etc. se pueden especificar páginas de error personalizadas. Para hacer esto, se debe modificar el archivo **Web.config** agregando en la sección **customErrors** el tag error, luego, en el atributo **statusCode**, se indica el código del error para el cual se tiene una página de error personalizada; finalmente, en el atributo **redirect** se indica la página de error personalizada que se mostrará al usuario cuando se tenga un error con el código especificado.

Se puede incluir el atributo **defaultRedirect** que especifica si la solicitud debe ser direccionada a una página específica ante la ocurrencia de cualquier error.

El siguiente ejemplo especifica que ante un código de **error 404** el usuario debe ser direccionado a la página **Error404.aspx** y si se genera cualquier otro código de error el usuario debe ser redireccionado hacia **Error.aspx**

```
<customErrors mode="RemoteOnly" defaultRedirect="Error.aspx">
  <error statusCode="404" redirect="Error404.aspx" />
</customErrors>
```

4. Seguimiento de la aplicación Web



El seguimiento es una manera de monitorear la ejecución de una aplicación ASP.NET, lo cual permite grabar los detalles de la excepción y flujo del programa, de manera que no afecte el resultado del mismo. En ASP.NET existe un soporte enriquecido para realizar el seguimiento, que permite configurar el destino de la salida mediante **TraceListeners**, tales como **EventLogTraceListener**, los que permiten guardar el seguimiento en el Registro de Eventos de Windows, al igual que los archivos **logs** e inclusive en otro tipo de formatos y hasta en una base de datos (**CustomTraceListeners**).

Seguimiento a nivel de página

Es habilitado añadiendo **"Trace=true"** a la directiva Page en cualquier página ASP.NET.

```
<%@ Page Language="vb" AutoEventWireup="False" Trace="True"
  Codebehind="TraceTest.aspx.vb" Inherits="TracingExample.Test"%>
```

Adicionalmente, se puede añadir el atributo **TraceMode** para establecer el método de ordenamiento del resultado del seguimiento por categoría (SortByCategory) o predeterminado (SortByTime). El ordenamiento **SortByTime** permite visualizar los métodos que toman mayor tiempo de

procesamiento en el CPU. Programáticamente, se puede habilitar el seguimiento estableciendo la propiedad **Trace.IsEnabled** a “True”.

Seguimiento a nivel de aplicación

Es posible configurar en un solo lugar el seguimiento y habilitarlo para toda la aplicación al añadir en el Web.config el tag de **trace** dentro del tag **system.web**. Este tag debe tener una atributo **enabled=”true”**.

```
<configuration>
  <system.web>
    <trace enabled="true" pageOutput="false" requestLimit="20" localOnly="false"/>
  </system.web>
</configuration>
```

Se pueden usar los atributos **pageOutput=”false”** y **requestLimit=”20”, que** son usados para indicar que se almacenen 20 peticiones de seguimiento pero que no se muestren en la página porque el atributo **pageOutput** está establecido como **false**.

Las configuraciones de seguimiento, a nivel de página, predominan sobre las del web.config, esto quiere decir que si **enabled=”false”** es establecido en web.config, pero en la página se configura **trace=”true”**, el seguimiento estará habilitado únicamente para esa página.

Visualización de datos del seguimiento

El seguimiento puede ser visualizado para múltiples solicitudes de página, a nivel de aplicación, invocando a una página especial denominada **trace.axd**. Cuando ASP.NET detecta una solicitud HTTP para trace.axd, la solicitud es atendida por la clase **TraceHandler** en lugar de una página.