CODERAGE 2025

December 1-5

10 am -4 pm (CST)

embarcadero®

Agenda

Interfaces: What to do and what not to do!

- Memory management
- Pitfalls
- Best practices

Speaker Background – Carlos Agnes

- Embarcadero MVP
- B.Sc. in Computer Science
- Several Delphi Certifications
- 25+ years of experience as a software developer
- 15+ years as a speaker at Delphi events

- CEO at TMR (please visit tmrti.com.br)
- carlos.agnes@tmrti.com.br

TMR

Interfaces 1.0.1

Interface definition and how it works

```
IOrder = interface
  ['{4363F2C3-3A49-4DC9-B422-8582D43FA494}']

  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;

  property CustomerID: UInt32 read GetCustomerID write SetCustomerID;
  property ItemsCount: NativeInt read GetItemsCount;
  property Item[const pIndex: NativeInt]: IOrderItem read GetItem;
end;
```

And that's how coupling starts…

```
IOrder = interface
  ['{4363F2C3-3A49-4DC9-B422-8582D43FA494}']

  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;

  function ExportToJSON: TJSONObject;
  procedure ImportFromDataset(const pDataset: TDataset);

  property CustomerID: UInt32 read GetCustomerID write SetCustomerID;
  property ItemsCount: NativeInt read GetItemsCount;
  property Item[const pIndex: NativeInt]: IOrderItem read GetItem;
end;
```

Keep it decoupled

```
IOrderJSONConverter = interface
  ['{ACFCFF91-98C0-418F-9D21-58F70BF8DEFC}']

  function Export(const pOrder: IOrder): TJSONObject;
end;

IOrderDatasetConverter = interface
  ['{409FC24F-286C-43CE-8D83-87645CB9CD2F}']

  procedure LoadOrder(const pOrder: IOrder; const pDataset: TDataset);
end;
```

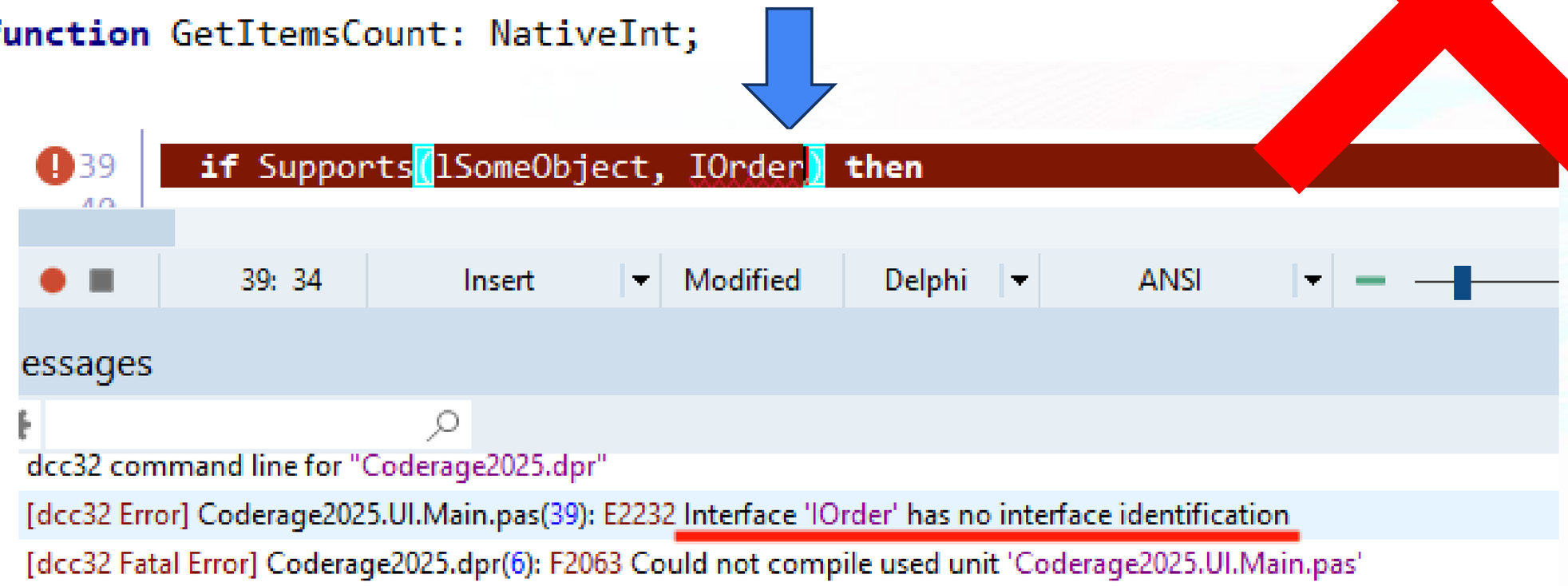
Delphi specifics

The interface GUID

```
IOrder = interface
  ['{4363F2C3-3A49-4DC9-B422-8582D43FA494}']

  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;

  property CustomerID: UInt32 read GetCustomerID write SetCustomerID;
  property ItemsCount: NativeInt read GetItemsCount;
  property Item[const pIndex: NativeInt]: IOrderItem read GetItem;
end;
```

If you don't declare a GUID to an Interface…

```
IOrder = interface
  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
```



- Use Ctrl+Shift+G to create a new GUID to your new interface.

IInterface inheritance and its methods

```
IInterface = interface
  ['{00000000-0000-0000-C000-000000000046}']
  function QueryInterface(const IID: TGUID; out Obj): HResult; stdcall;
  function _AddRef: Integer; stdcall;
  function _Release: Integer; stdcall;
end;
```

TInterfacedObject

```
TOrder = class(TInterfacedObject, IOrder)
strict private
  FCustomerID: UInt32;
  FItems: TList<IOrderItem>;
strict protected
  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;
public
  constructor Create;
  destructor Destroy; override;
end;
```

Don’t use a TInterfacedObject this way

```
procedure ProccessOrder(pOrder: IOrder);

var lOrder := TOrder.Create;

try
  // some code involving the order object here

  ProccessOrder(lOrder);
finally
  lOrder.Free;
end;
```

The right way to use a TInterfacedObject

```
var lOrder: IOrder := TOrder.Create;

// some code involving the order object here

ProccessOrder(lOrder);
```

What if you don't want ARC on your interfaced class?

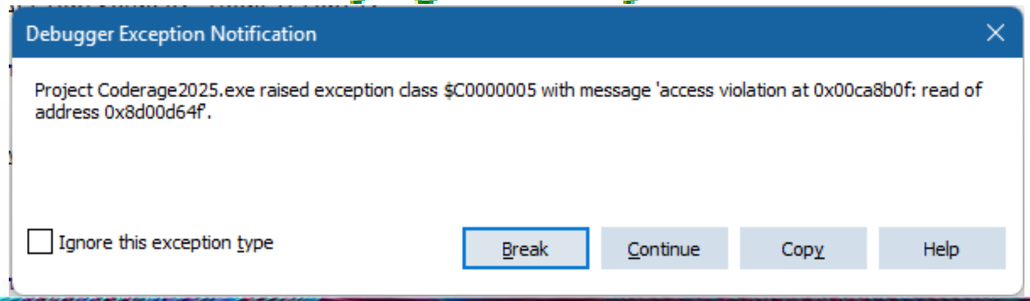
```
TOrder = class(TComponent, IOrder)
strict private
  FCustomerID: UInt32;
  FItems: TList<IOrderItem>;
strict protected
  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;
public
  constructor Create;
  destructor Destroy; override;
end;
```

The best way to avoid ARC

```
TOrder = class(TNoRefCountObject, IOrder)
strict private
  FCustomerID: UInt32;
  FItems: TList<IOrderItem>;
strict protected
  function GetCustomerID: UInt32;
  procedure SetCustomerID(const pID: UInt32);

  function GetItemsCount: NativeInt;
  function GetItem(const pIndex: NativeInt): IOrderItem;
public
  constructor Create;
  destructor Destroy; override;
end;
```

Even non-ARC classes can cause headaches

```
var
  lSomeDialog: ISomeDialog;
begin
  lSomeDialog := TSomeForm.Create;

  lSomeDialog.Setup;

  lSomeDialog.Execute; // shows the dialog as a modal window
                       // but it doesn't stop there...
                       // this dialog opens another dialog
                       // and the TSomeForm instance realeses itself
                       // because it is configured by its close action
end;
```



Memory management attributes

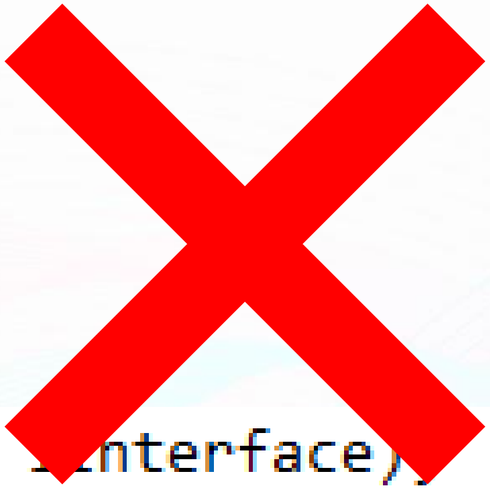
- weak
- unsafe

```
var
  [unsafe] lSomeDialog: ISomeDialog;
begin
  lSomeDialog := TSomeForm.Create;

  lSomeDialog.Setup;

  lSomeDialog.Execute; // shows the dialog as a modal window
                       // but it doesn't stop there...
                       // this dialog opens another dialog
                       // and the TSomeForm instance realeses itself
                       // because it is configured by its close action

end;
```

Circular references

```
TOrderItem = class(TInterfacedObject, IOrderItem)
strict private
  [weak] FOrder: IOrder;

TOrder = class(TInterfacedObject, IOrder)
strict private
  FCustomerID: UInt32;
  FItems: TList<IOrderItem>;
```

Unsafe/weak x Supports



```
procedure TCodeRage2025.TestSupport(pInterface: IInterface);
var
  [weak] lSomeInterface: ISomeInterface;
begin
  if Supports(pInterface, ISomeInterface, lSomeInterface) then
    { ... }
end;
```

Interface inheritance

Q: Is there a magic number for the maximum number of interface inheritance levels?

A: As long as the inheritance hierarchy remains coherent and meaningful, I see no reason to enforce a limit.

Interfaces and UI – how to do it properly?

```
IView = interface
  ['{6CC3CCE4-E6BD-438E-BF32-AB4E2F6E0645}']

  procedure Show;
end;

TFMXForm = class(TForm)
private
  { Private declarations }
public
  { Public declarations }
end;

TFMXView<F: TFMXForm> = class(TInterfacedObject, IView)
strict private
  Form: F;
strict protected
  procedure Show;
```

Interfaces X Generics in Methods

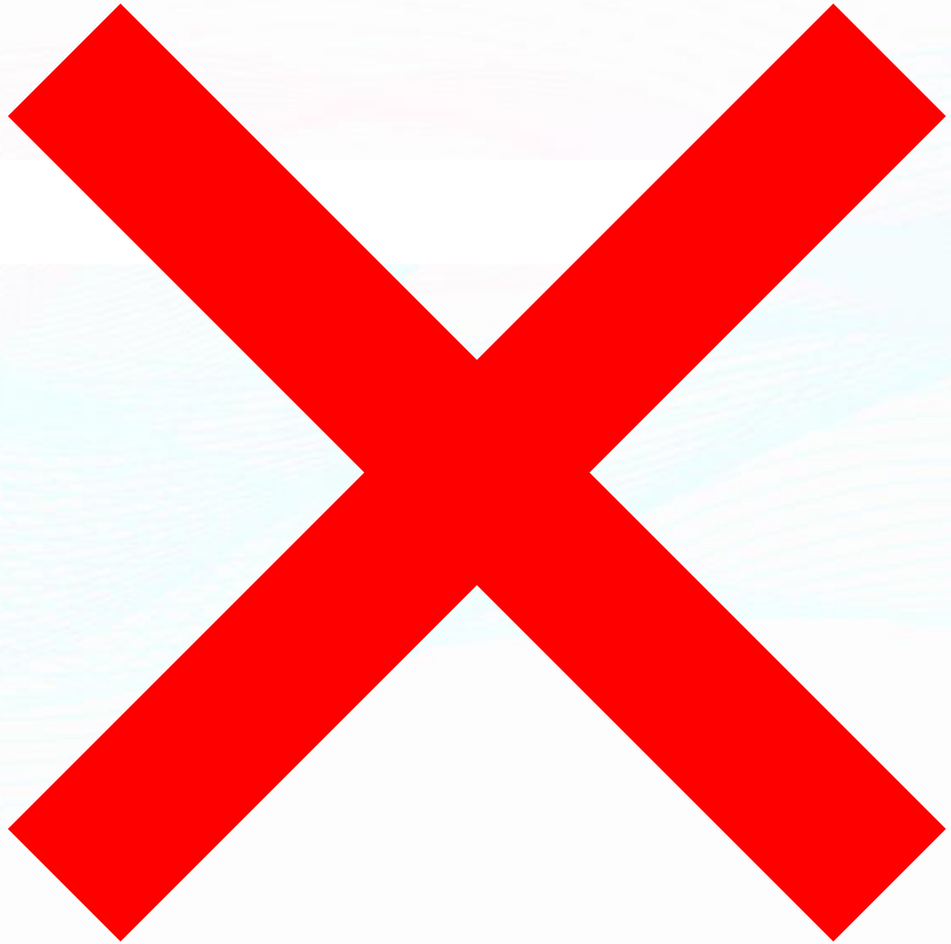
```
constructor TOrderDomain.Create(pDIContainer: IDIContainer);
begin
  inherited Create;

  FOrderRepository := pDIContainer.Implementations.Get<IOrderRepository>;
end;
```

Interface Object Method with Generics

The final Don’t!

- Please, never, I said never, typecast the interface back to its implementation class type!

```
var lOrderObject := lOrder as TOrder;
```



Thank You!

carlos.agnes@tmrti.com.br