Embarcadero Conference 2023

Como usar padrões de projeto e testes unitários para criar sistemas de alta qualidade

{ *Palestrante*

Cesar Romero

Cesar Romero

Software Architect

@cesarliws

#delphers

Clean Architecture

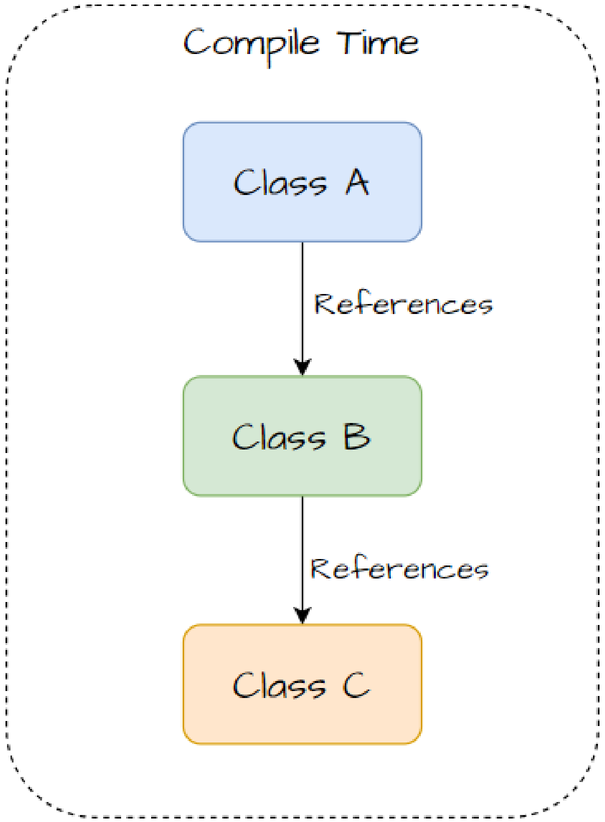
Make what is right easy and what is wrong difficult.

Steve "Ardalis" Smith

Agenda

- Introdução
- Conceitos básicos
- Benefícios
- Desafios
- Exemplo Prático
- Conclusão

Gráfico de Dependência Direta



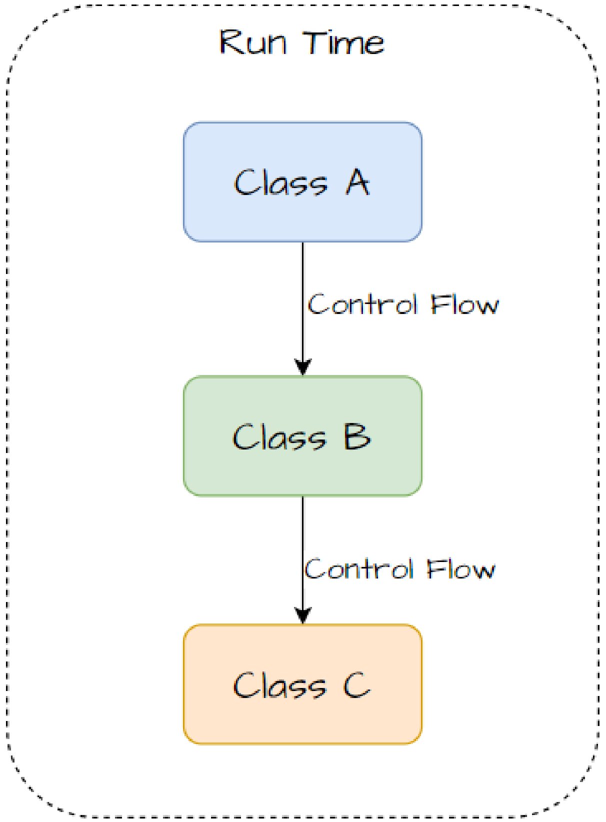
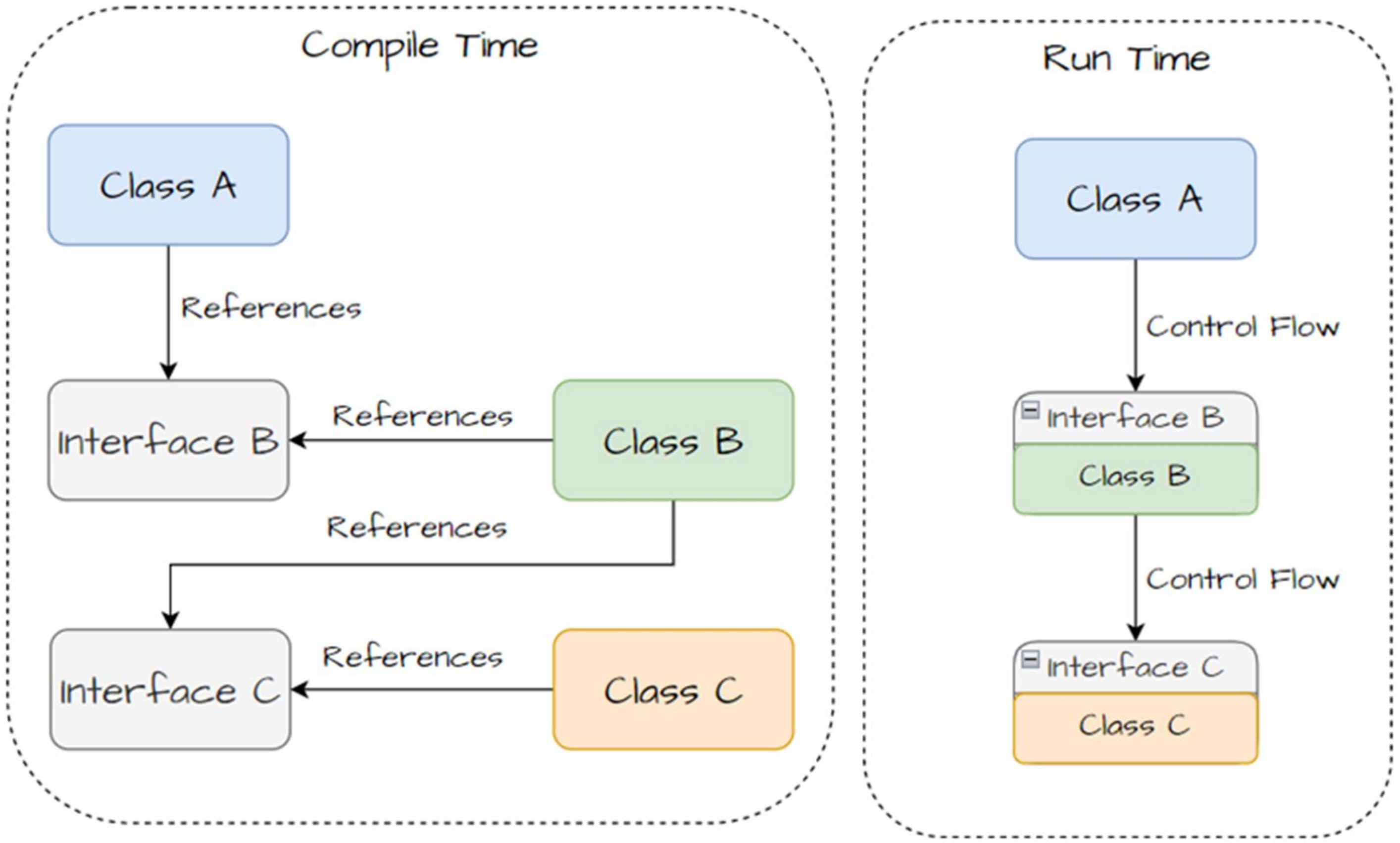
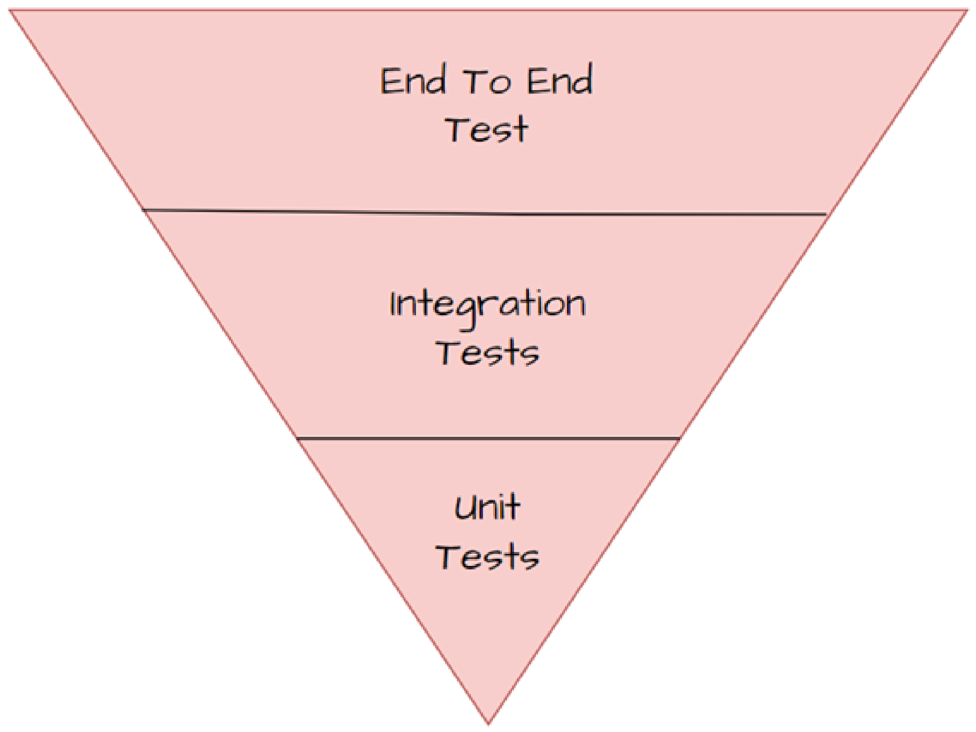


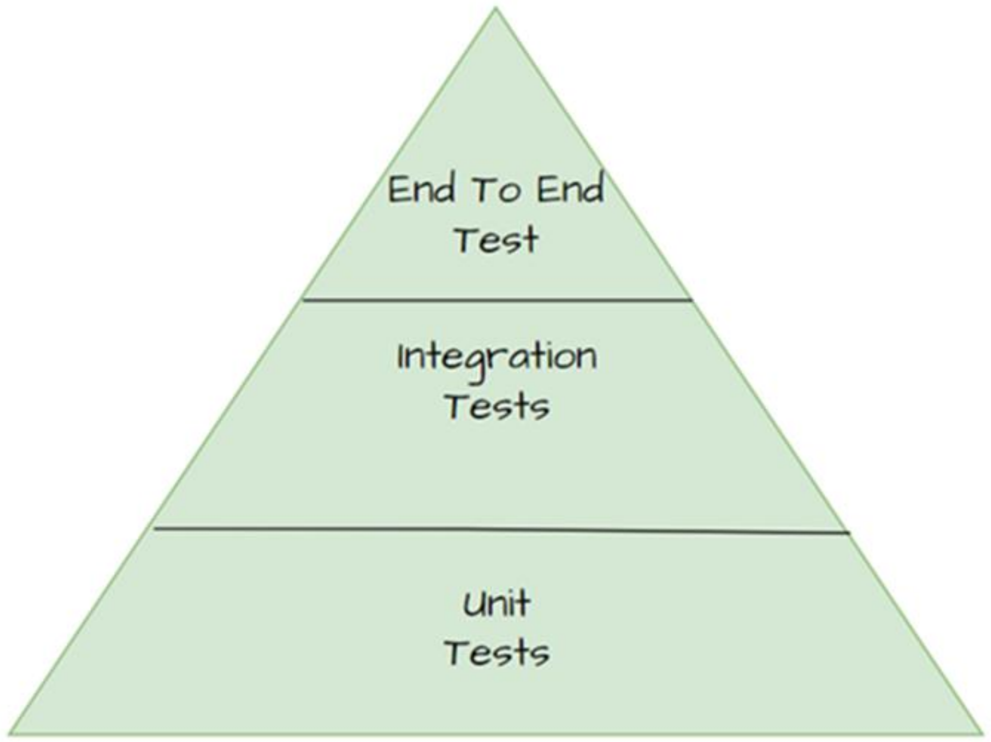
Gráfico de Dependência Invertida

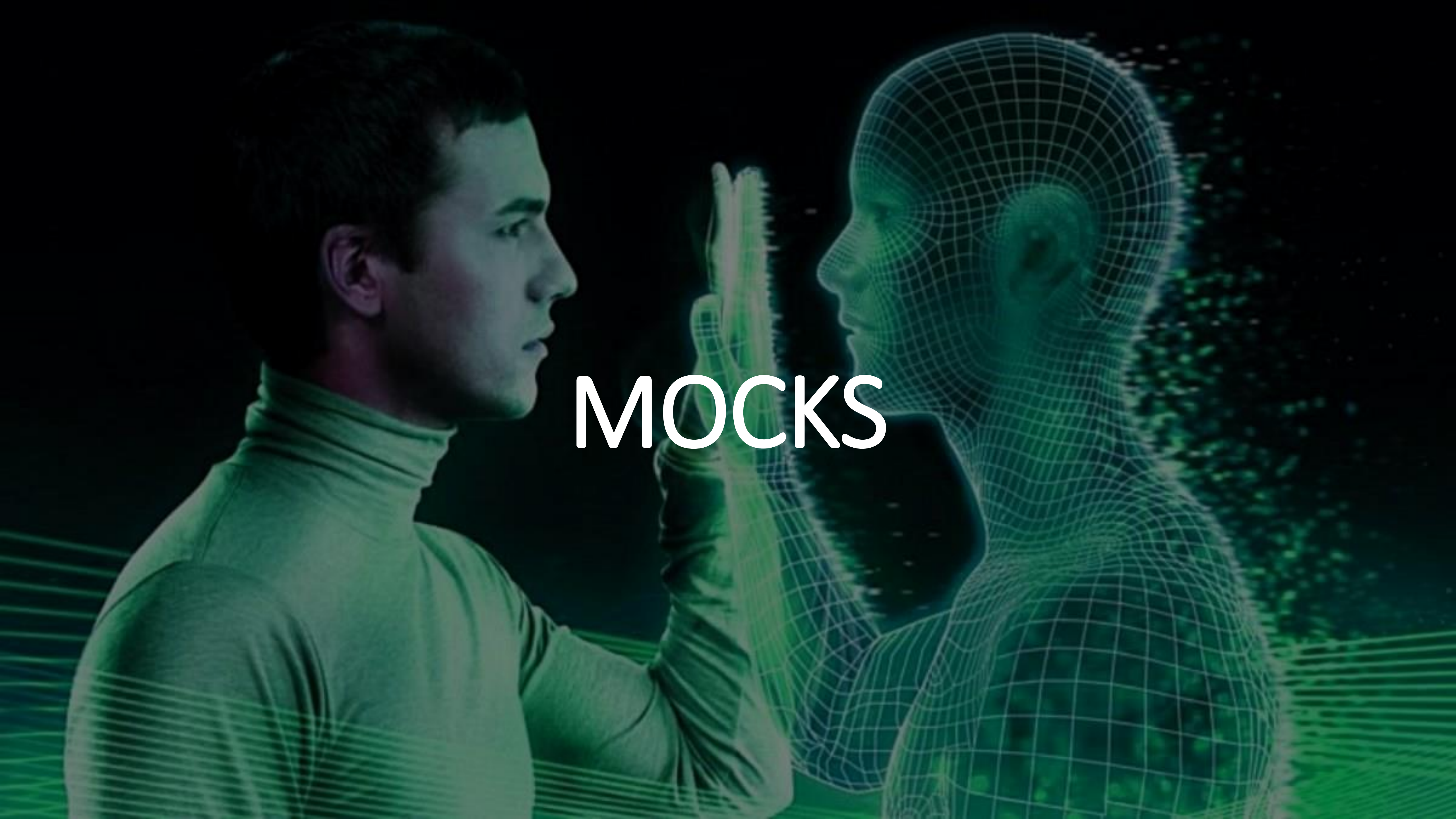


Pirâmide Invertida de Testes



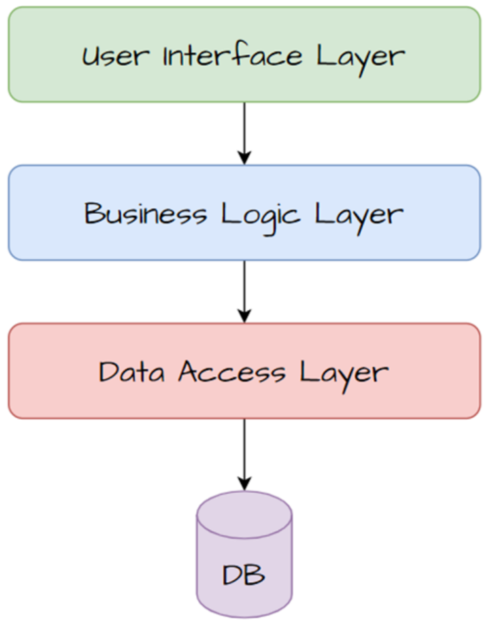
Pirâmide de Testes



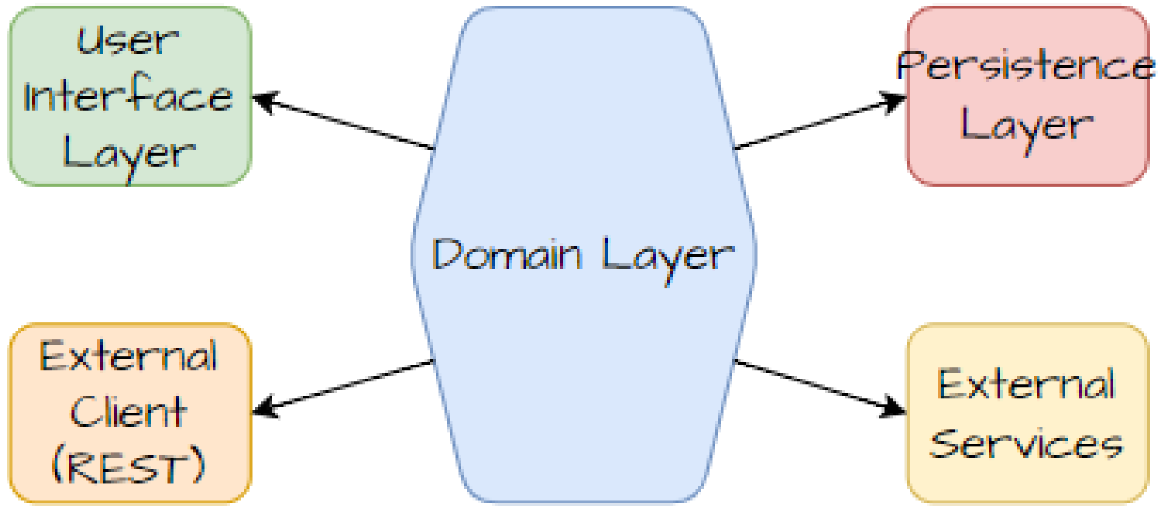


MOCKS

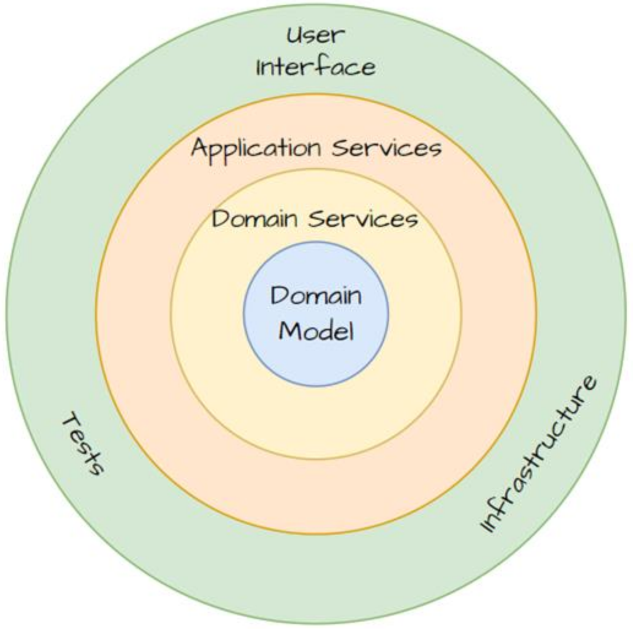
Arquitetura de Camadas Clássica



Hexagonal Architecture

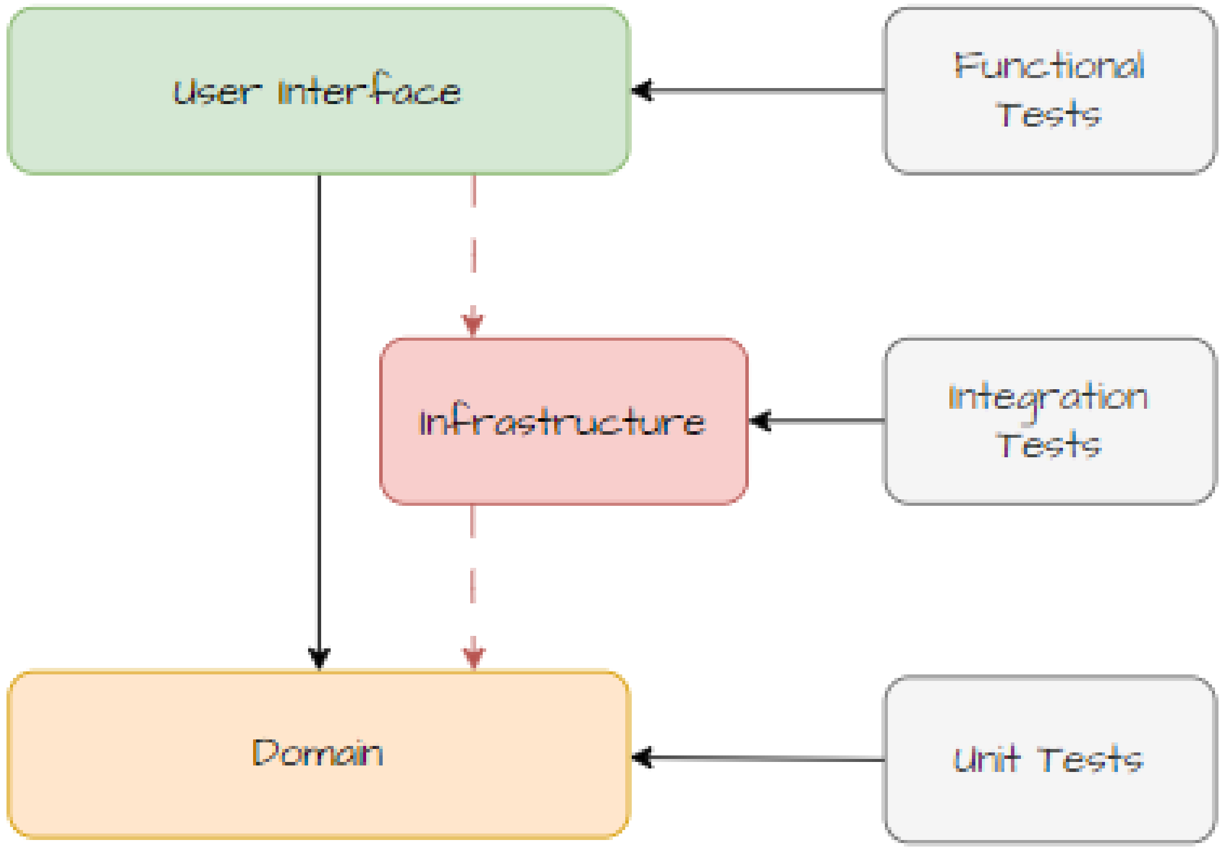


Onion Architecture



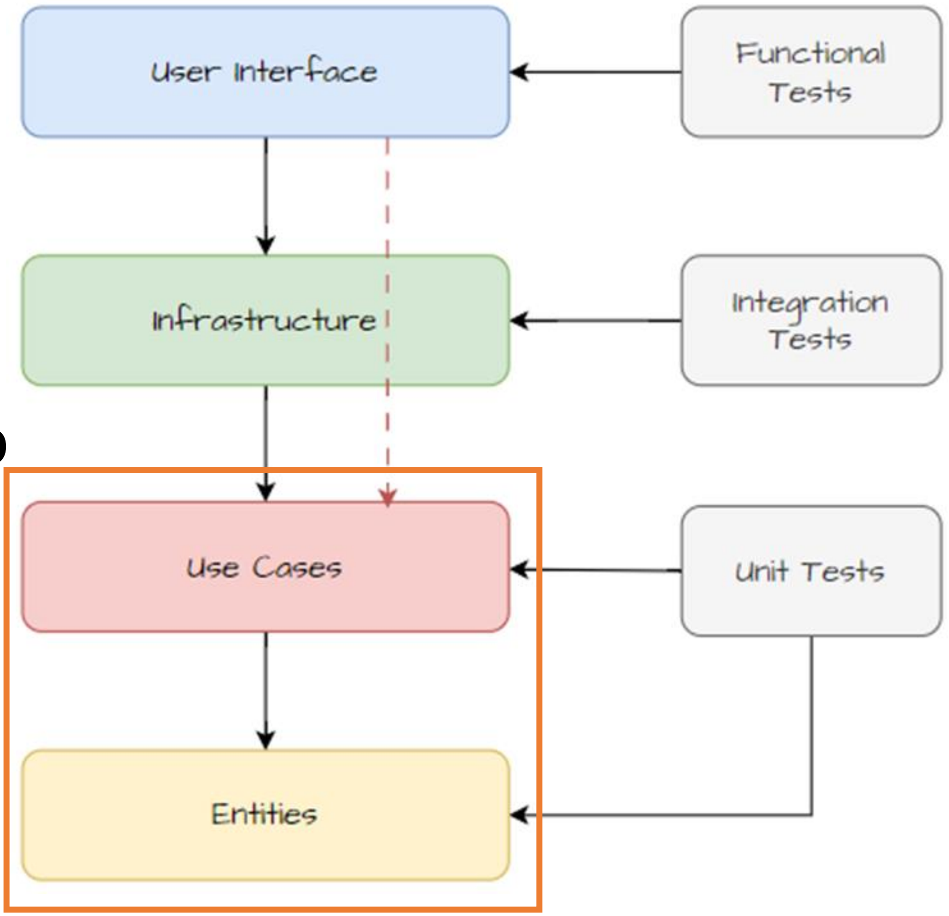
Clean Architecture

- Independência da UI
- Independência de DB
- Independência de Agentes Ex
- Testabilidade

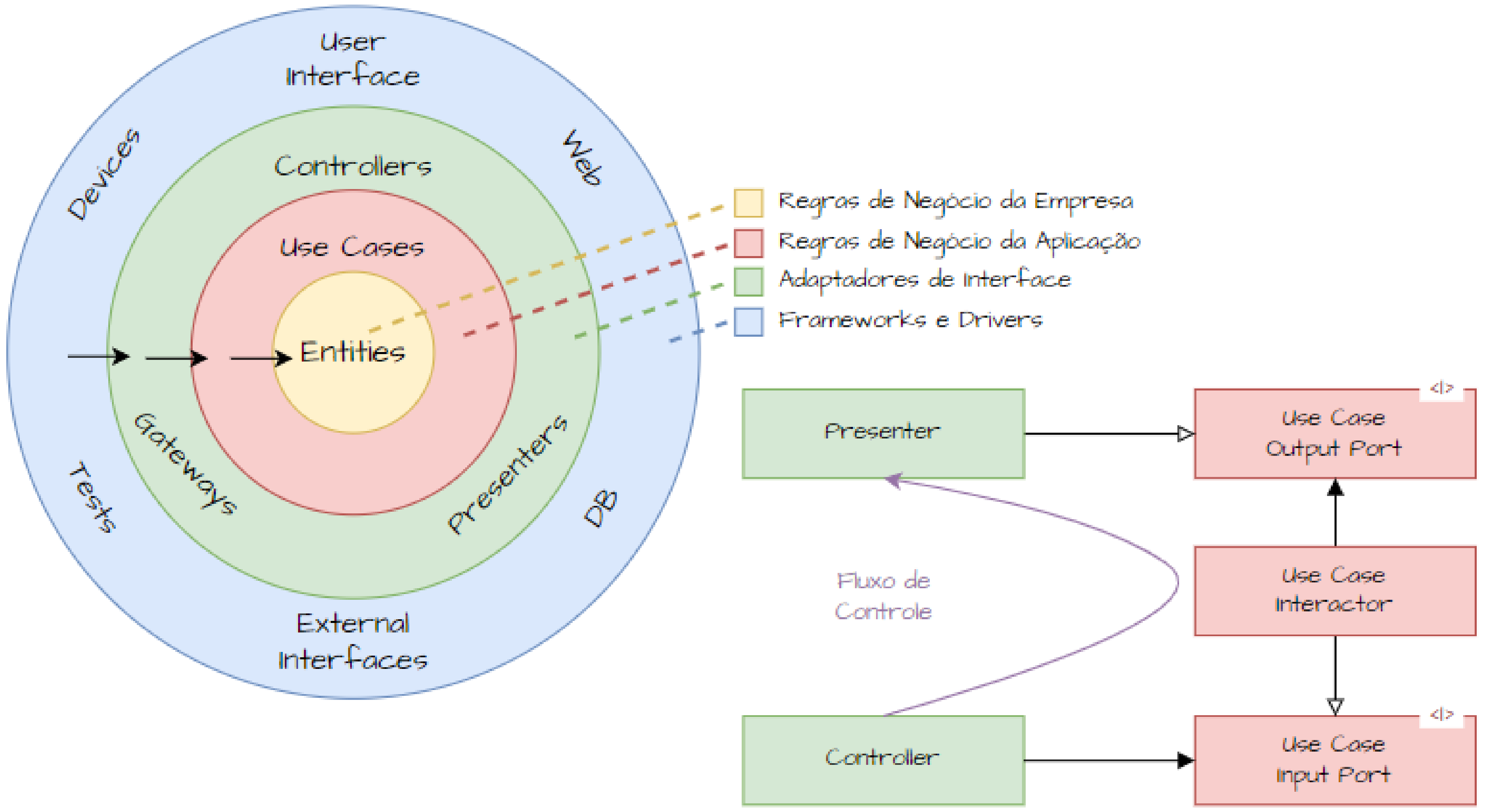


Clean Architecture

- Independência da UI
- Independência de DB
- Independência de Agentes Externo
- Testabilidade



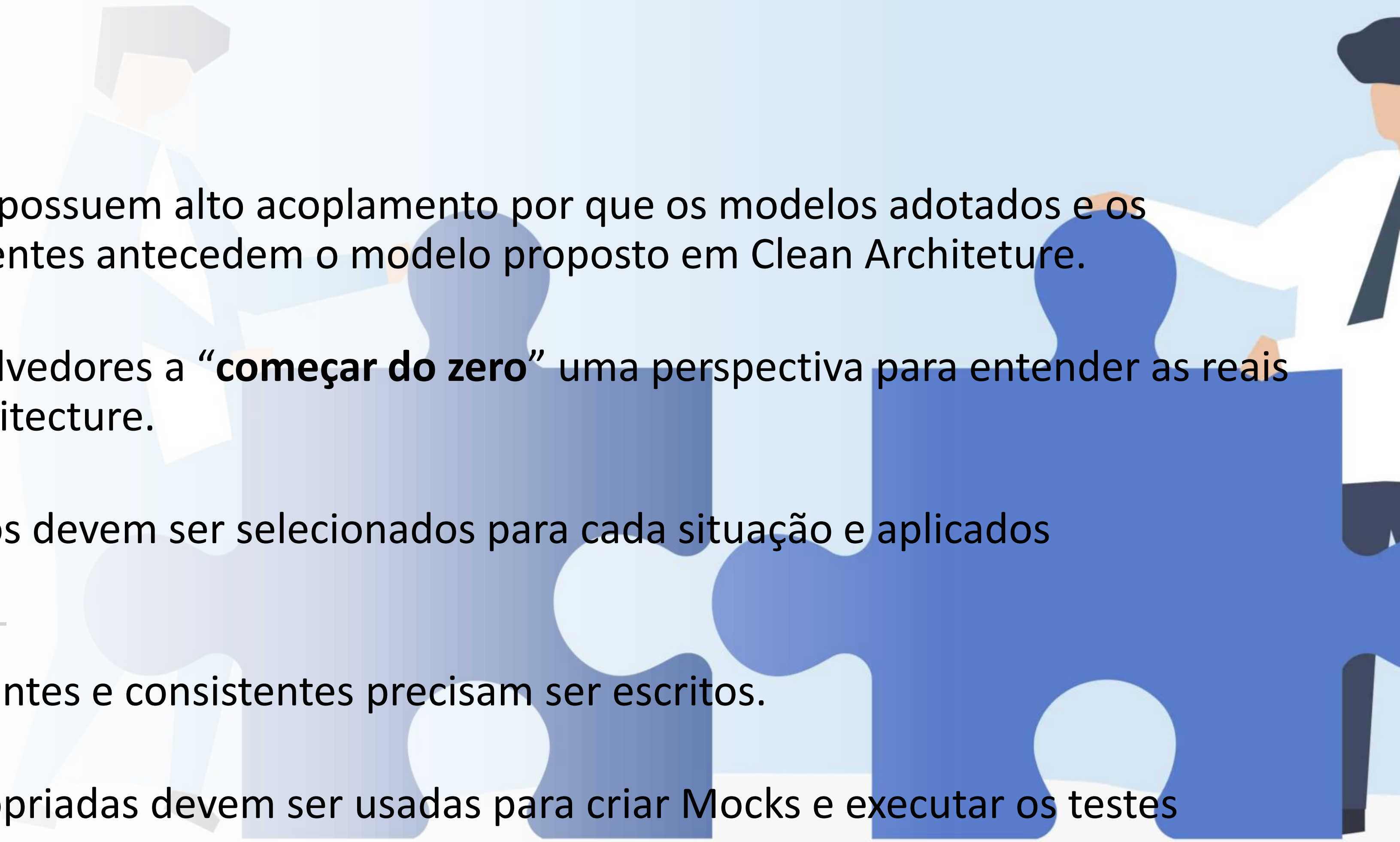
Clean Architecture



Benefícios

- Melhorar a qualidade e manutenção do código
- Reduzir o acoplamento e aumentar a coesão das classes
- Facilitar a descoberta e correção de Erros
- Aumentar a confiança e a produtividade dos desenvolvedores

Desafios



- Projetos em Delphi geralmente possuem alto acoplamento por que os modelos adotados e os templates Delphi com Componentes antecedem o modelo proposto em Clean Architeture.
- É preciso convencer os desenvolvedores a “**começar do zero**” uma perspectiva para entender as reais vantagens do uso de Clean Architecture.
- **Padrões de projetos** apropriados devem ser selecionados para cada situação e aplicados corretamente.
- **Testes Unitários** claros, abrangentes e consistentes precisam ser escritos.
- Frameworks e ferramentas apropriadas devem ser usadas para criar Mocks e executar os testes unitários.

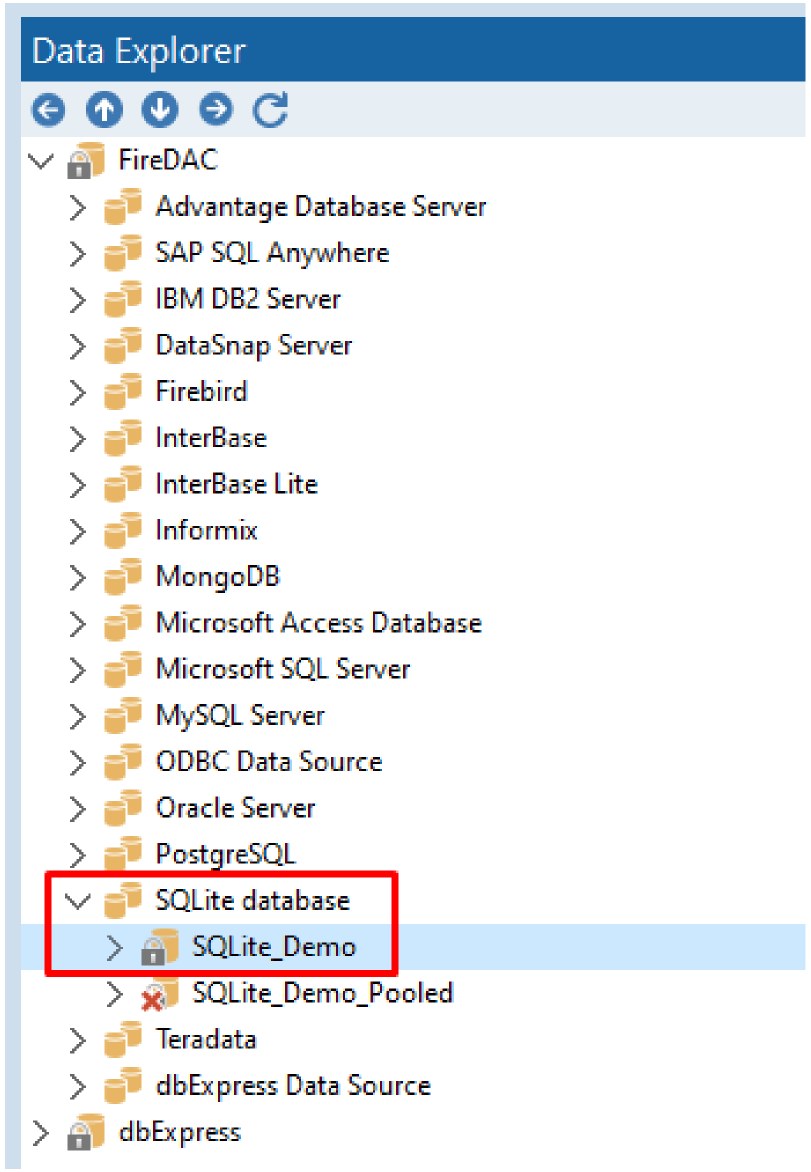
Exemplo Prático

Repositório

github.com/cesarliws/clean_architecture_delphi_bootcamp_2023

Dependências

- Delphi 11 Version 28.0.47991.2819
- Database Sqlite_demo - FireDAC

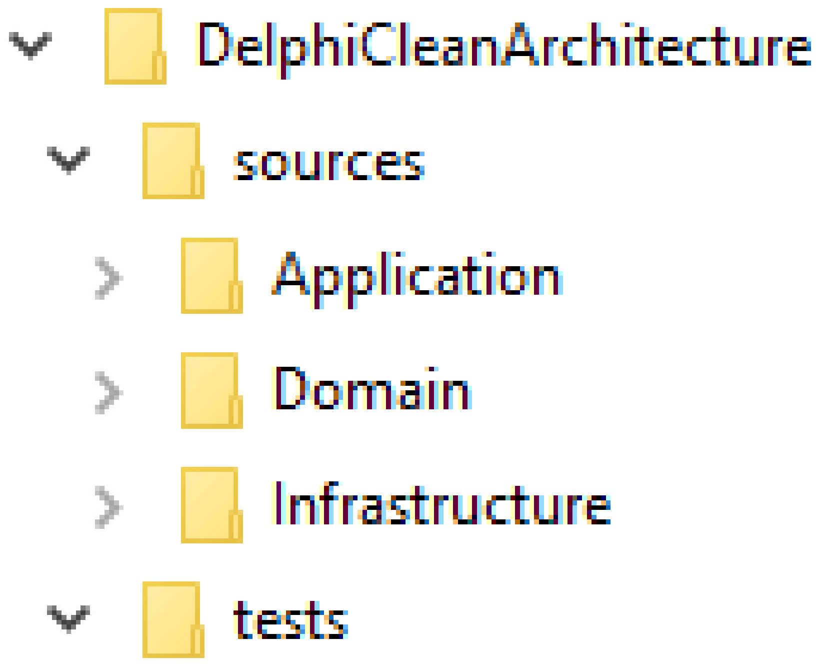

Data Explorer

- FireDAC
  - Advantage Database Server
  - SAP SQL Anywhere
  - IBM DB2 Server
  - DataSnap Server
  - Firebird
  - InterBase
  - InterBase Lite
  - Informix
  - MongoDB
  - Microsoft Access Database
  - Microsoft SQL Server
  - MySQL Server
  - ODBC Data Source
  - Oracle Server
  - PostgreSQL
  - SQLite database
    - SQLite_Demo
    - SQLite_Demo_Pooled
  - Teradata
  - dbExpress Data Source
- dbExpress

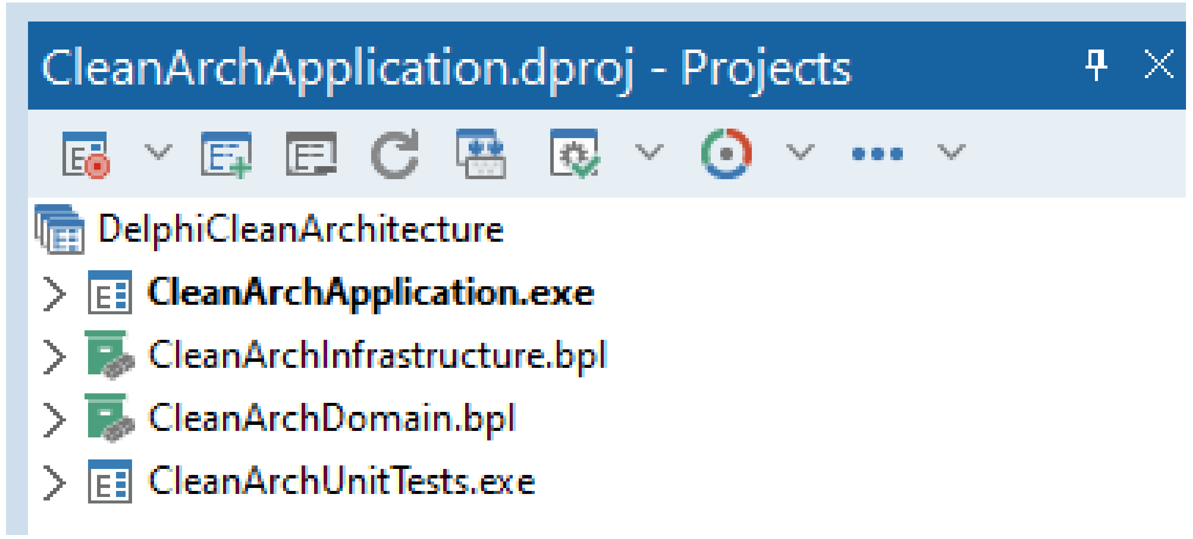
Dependências

- Dependency Injection
- Collections
- Persistence
- Mocks

Organização do Projeto



DelphiCleanArchitecture
sources
Application
Domain
Infrastructure
tests

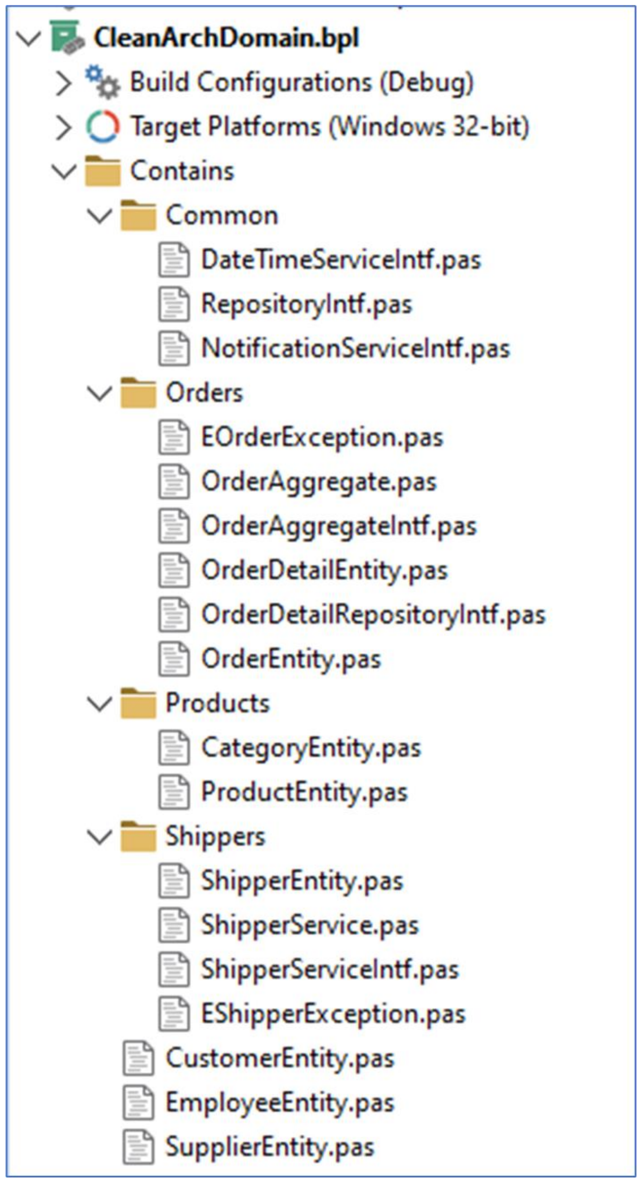


CleanArchApplication.dproj - Projects

DelphiCleanArchitecture
CleanArchApplication.exe
CleanArchInfrastructure.bpl
CleanArchDomain.bpl
CleanArchUnitTests.exe

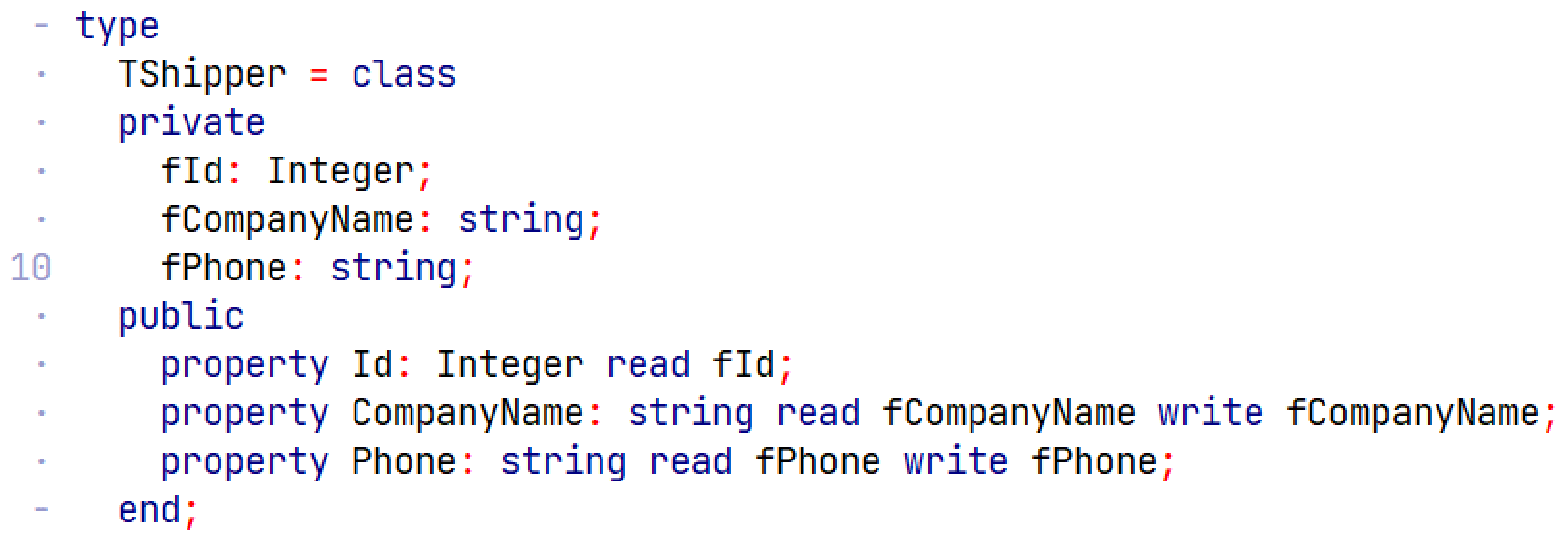
Projeto “Domain”

- Entities
- Aggregates
- Use Cases
- Exceptions
- Interfaces
- Domain Events



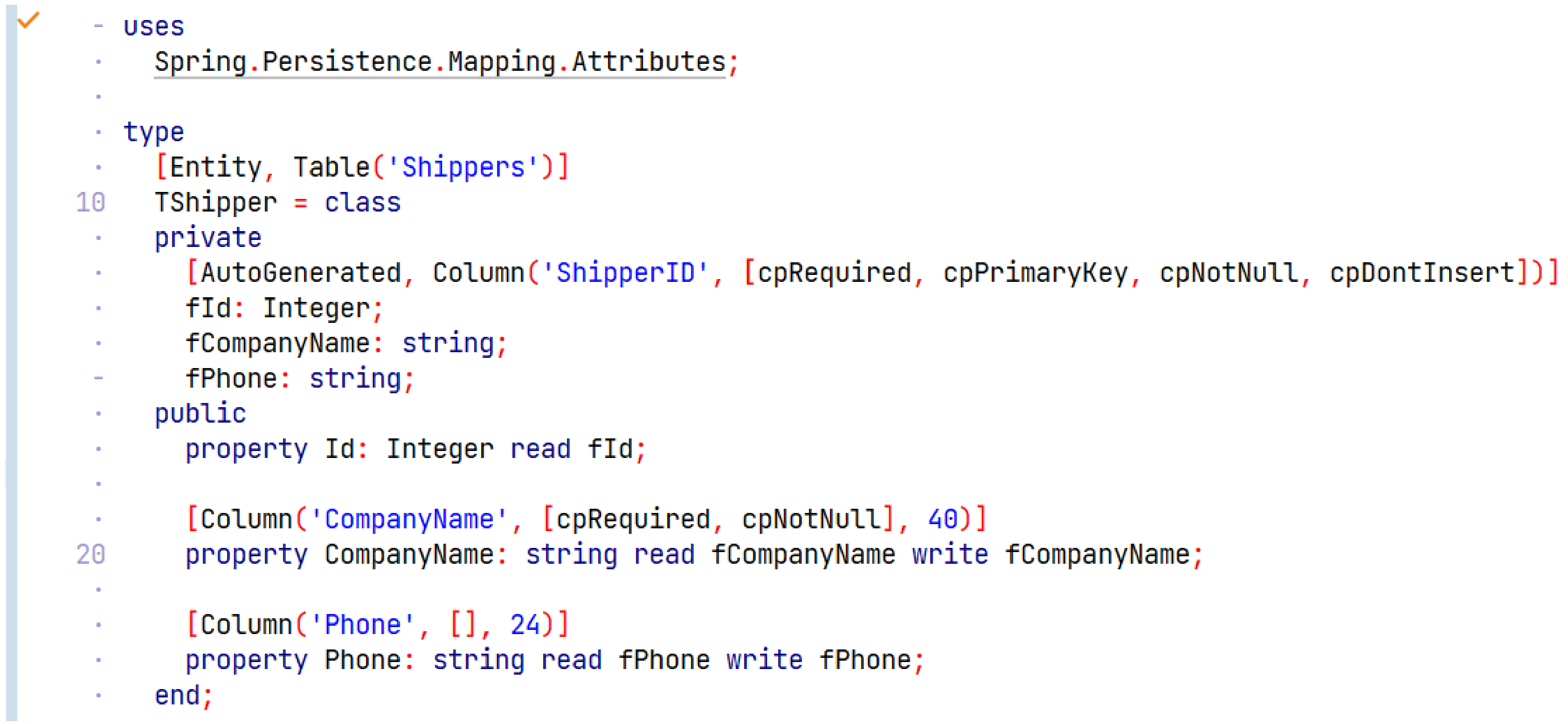
```
CleanArchDomain.bpl
  Build Configurations (Debug)
  Target Platforms (Windows 32-bit)
  Contains
    Common
      DateTimeServiceIntf.pas
      RepositoryIntf.pas
      NotificationServiceIntf.pas
    Orders
      EOrderException.pas
      OrderAggregate.pas
      OrderAggregateIntf.pas
      OrderDetailEntity.pas
      OrderDetailRepositoryIntf.pas
      OrderEntity.pas
    Products
      CategoryEntity.pas
      ProductEntity.pas
    Shippers
      ShipperEntity.pas
      ShipperService.pas
      ShipperServiceIntf.pas
      EShipperException.pas
    CustomerEntity.pas
    EmployeeEntity.pas
    SupplierEntity.pas
```

Entidade



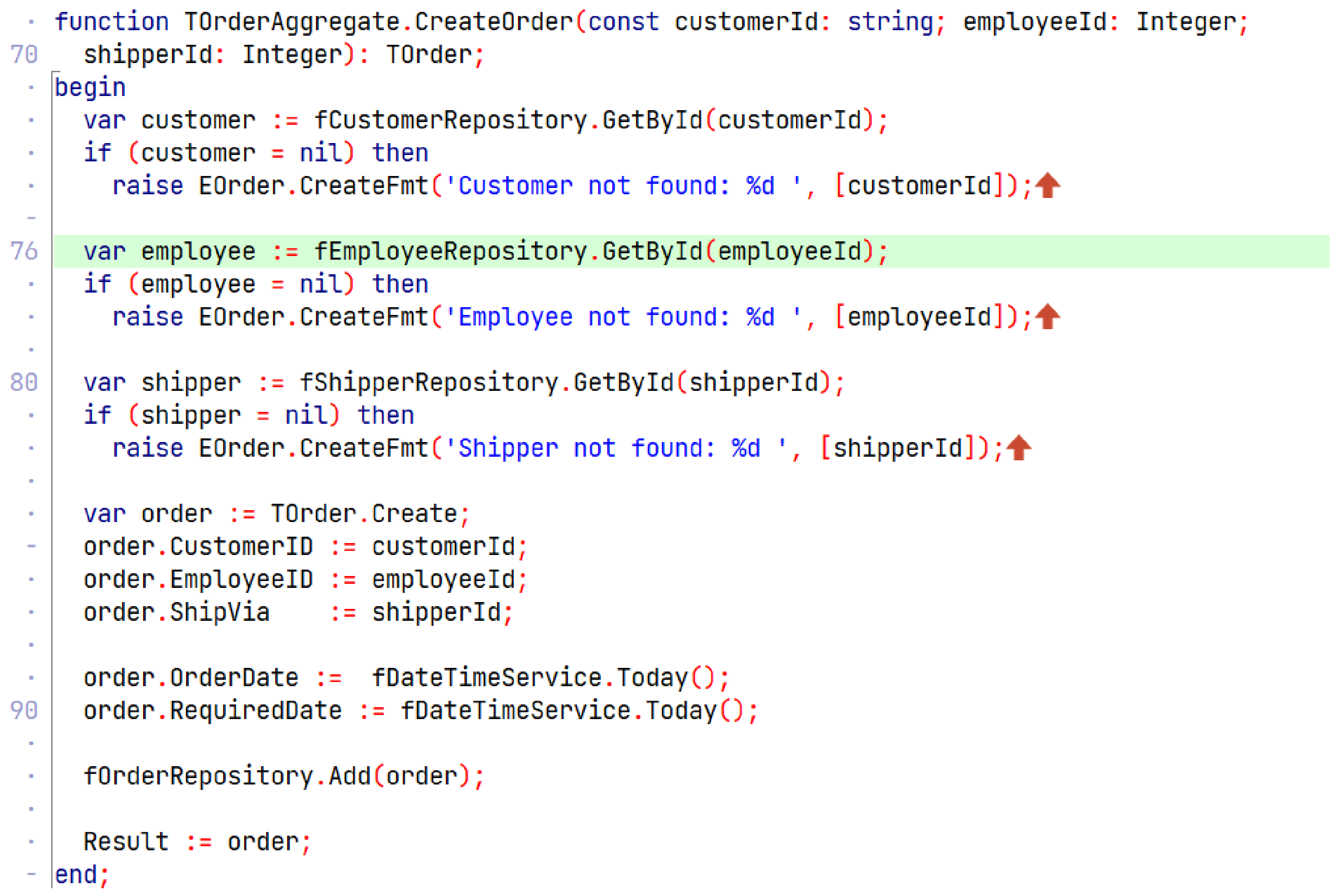
```
type
  TShipper = class
  private
    fId: Integer;
    fCompanyName: string;
    fPhone: string;
  public
    property Id: Integer read fId;
    property CompanyName: string read fCompanyName write fCompanyName;
    property Phone: string read fPhone write fPhone;
  end;
```

Entidade

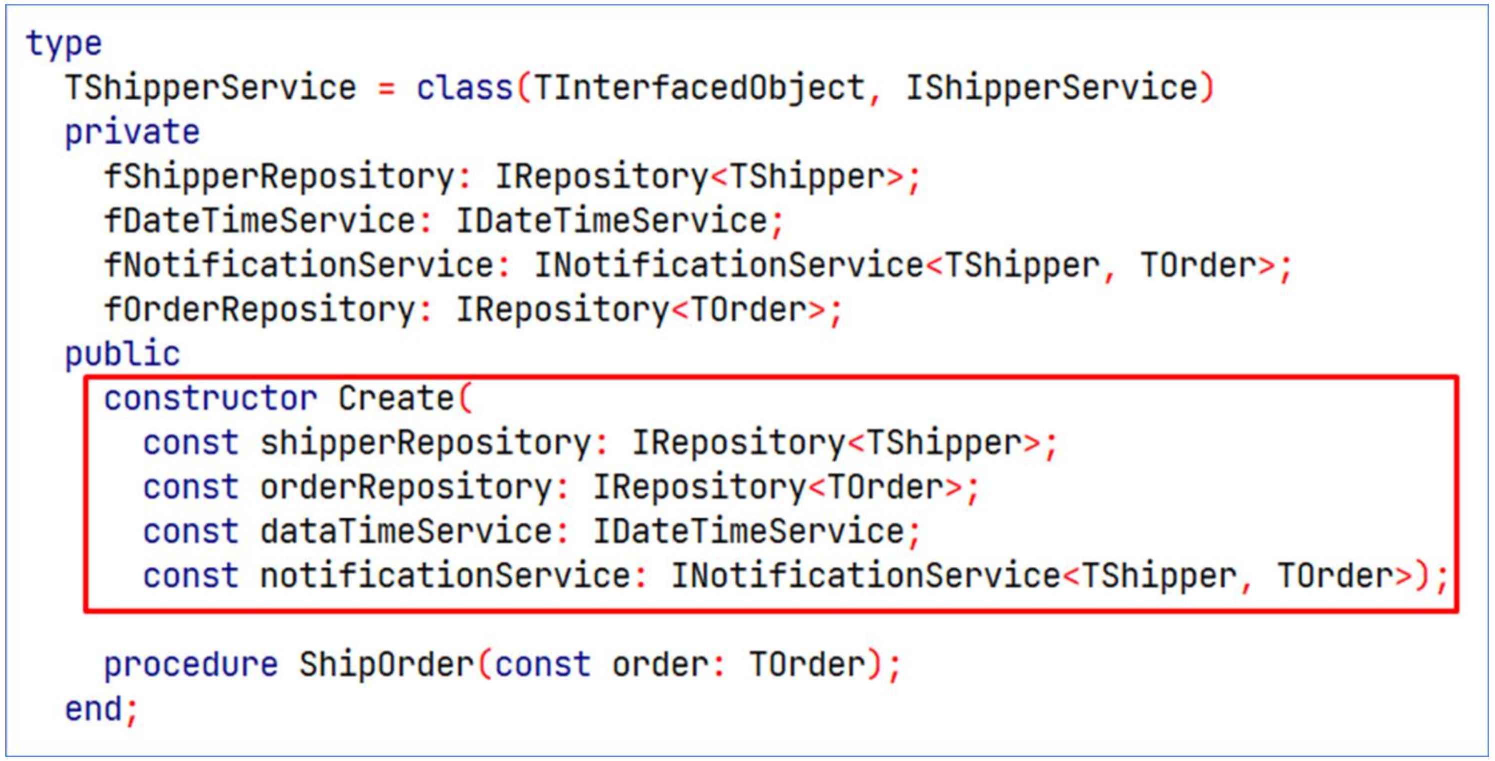


```
uses
  Spring.Persistence.Mapping.Attributes;

type
  [Entity, Table('Shippers')]
  TShipper = class
  private
    [AutoGenerated, Column('ShipperID', [cpRequired, cpPrimaryKey, cpNotNull, cpDontInsert])]
    fId: Integer;
    fCompanyName: string;
    fPhone: string;
  public
    property Id: Integer read fId;

    [Column('CompanyName', [cpRequired, cpNotNull], 40)]
    property CompanyName: string read fCompanyName write fCompanyName;

    [Column('Phone', [], 24)]
    property Phone: string read fPhone write fPhone;
  end;
```

Aggregate

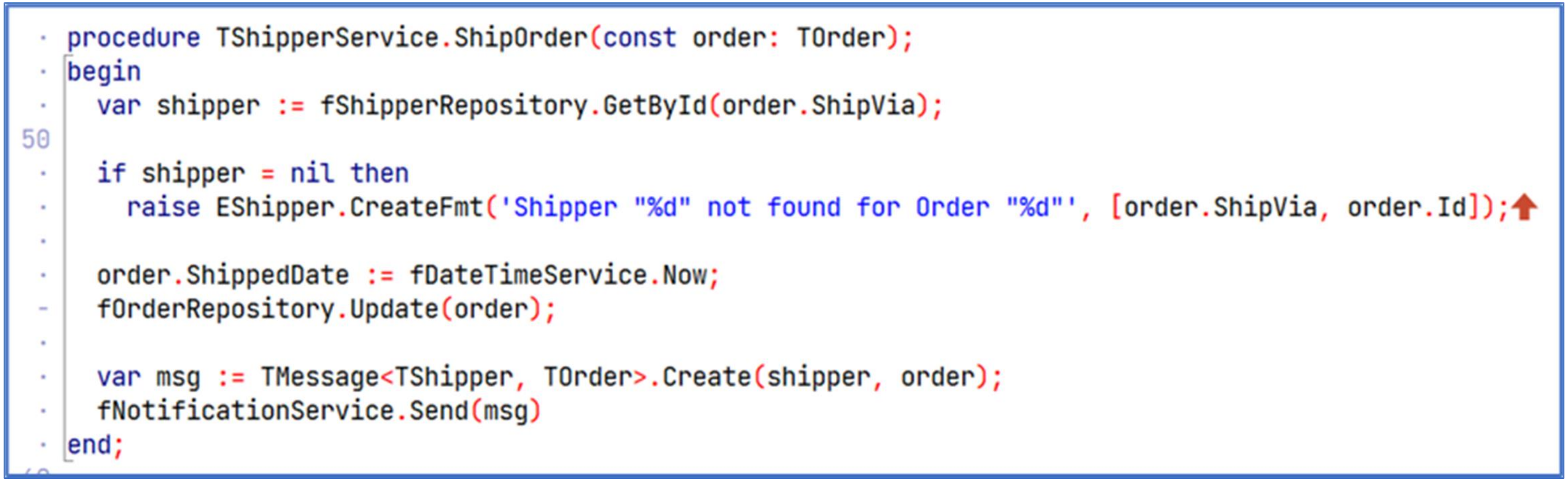

```
function TOrderAggregate.CreateOrder(const customerId: string; employeeId: Integer;
  shipperId: Integer): TOrder;
begin
  var customer := fCustomerRepository.GetById(customerId);
  if (customer = nil) then
    raise EOrder.CreateFmt('Customer not found: %d ', [customerId]);

  var employee := fEmployeeRepository.GetById(employeeId);
  if (employee = nil) then
    raise EOrder.CreateFmt('Employee not found: %d ', [employeeId]);

  var shipper := fShipperRepository.GetById(shipperId);
  if (shipper = nil) then
    raise EOrder.CreateFmt('Shipper not found: %d ', [shipperId]);

  var order := TOrder.Create;
  order.CustomerID := customerId;
  order.EmployeeID := employeeId;
  order.ShipVia    := shipperId;

  order.OrderDate :=  fDateTimeService.Today();
  order.RequiredDate := fDateTimeService.Today();

  fOrderRepository.Add(order);

  Result := order;
end;
```

Use cases



```
type
  TShipperService = class(TInterfacedObject, IShipperService)
  private
    fShipperRepository: IRepository<TShipper>;
    fDateTimeService: IDateTimeService;
    fNotificationService: INotificationService<TShipper, TOrder>;
    fOrderRepository: IRepository<TOrder>;
  public
    constructor Create(
      const shipperRepository: IRepository<TShipper>;
      const orderRepository: IRepository<TOrder>;
      const dataTimeService: IDateTimeService;
      const notificationService: INotificationService<TShipper, TOrder>);

    procedure ShipOrder(const order: TOrder);
  end;
```

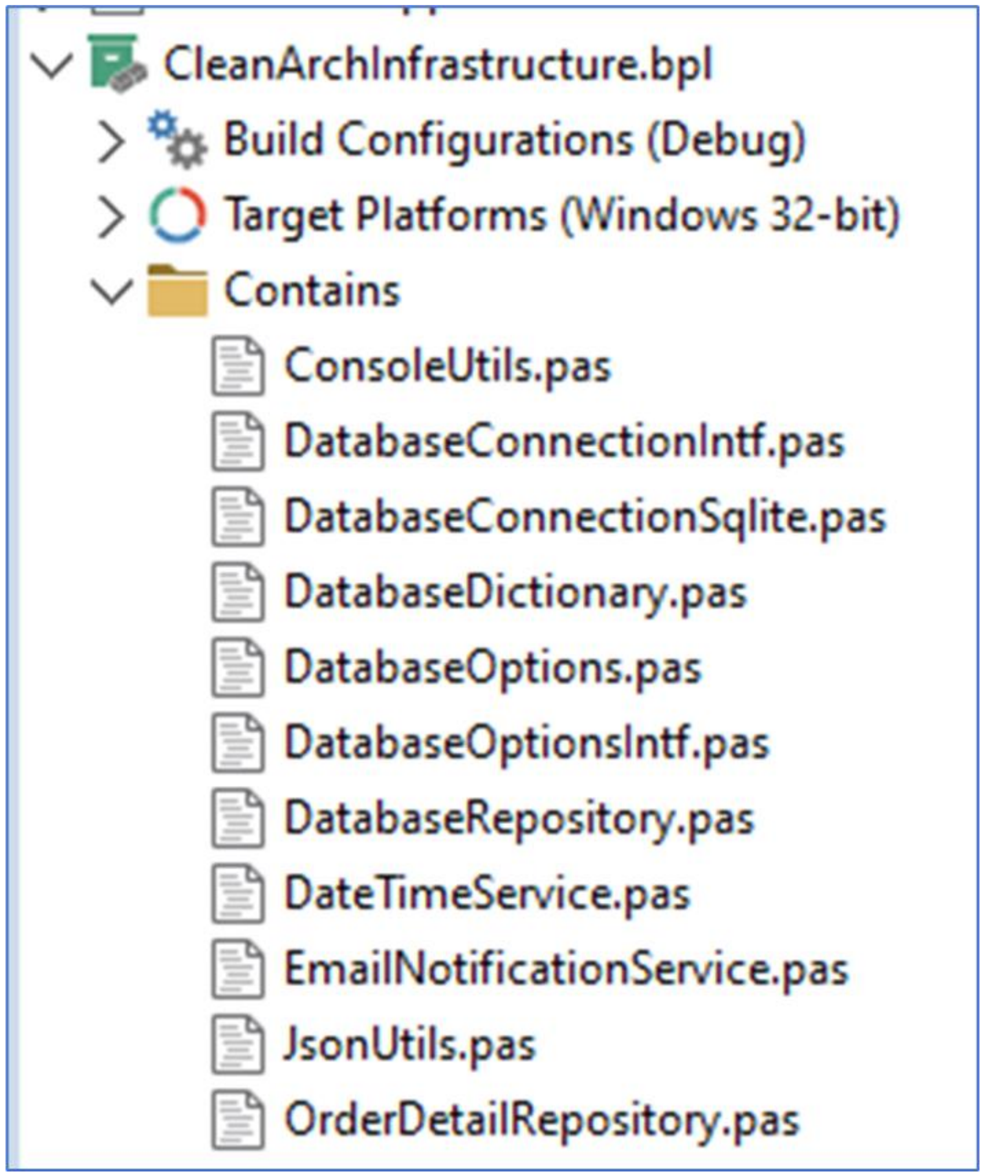
Use cases



```
procedure TShipperService.ShipOrder(const order: TOrder);
begin
  var shipper := fShipperRepository.GetById(order.ShipVia);

  if shipper = nil then
    raise EShipper.CreateFmt('Shipper "%d" not found for Order "%d"', [order.ShipVia, order.Id]);

  order.ShippedDate := fDateTimeService.Now;
  fOrderRepository.Update(order);

  var msg := TMessage<TShipper, TOrder>.Create(shipper, order);
  fNotificationService.Send(msg)
end;
```

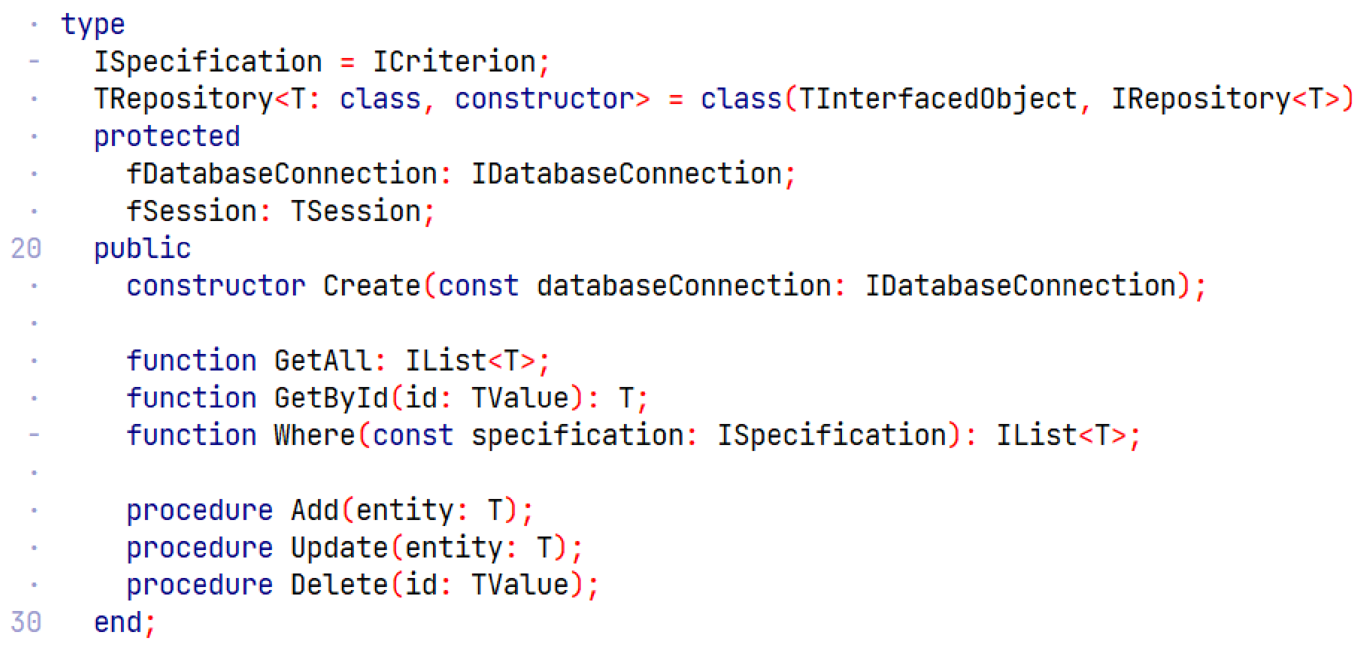
Projeto “Infraestructure”

- Dependências Externas
- DB
- Web
- IO
- Dispositivos



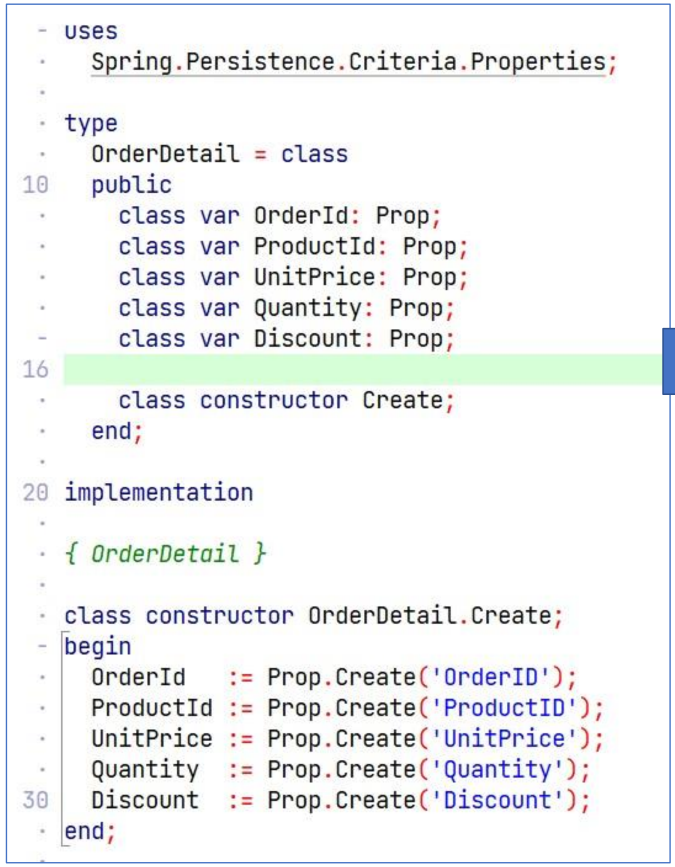
```
CleanArchInfrastructure.bpl
  Build Configurations (Debug)
  Target Platforms (Windows 32-bit)
  Contains
    ConsoleUtils.pas
    DatabaseConnectionIntf.pas
    DatabaseConnectionSqlite.pas
    DatabaseDictionary.pas
    DatabaseOptions.pas
    DatabaseOptionsIntf.pas
    DatabaseRepository.pas
    DateTimeService.pas
    EmailNotificationService.pas
    JsonUtils.pas
    OrderDetailRepository.pas
```

Padrão “Repository”

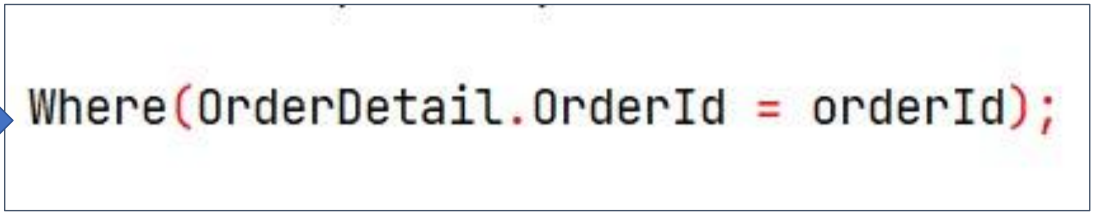


```
type
  ISpecification = ICriterion;
  TRepository<T: class, constructor> = class(TInterfacedObject, IRepository<T>)
  protected
    fDatabaseConnection: IDatabaseConnection;
    fSession: TSession;
  public
    constructor Create(const databaseConnection: IDatabaseConnection);

    function GetAll: IList<T>;
    function GetById(id: TValue): T;
    function Where(const specification: ISpecification): IList<T>;

    procedure Add(entity: T);
    procedure Update(entity: T);
    procedure Delete(id: TValue);
  end;
```

Padrão “Specification”em Consultas

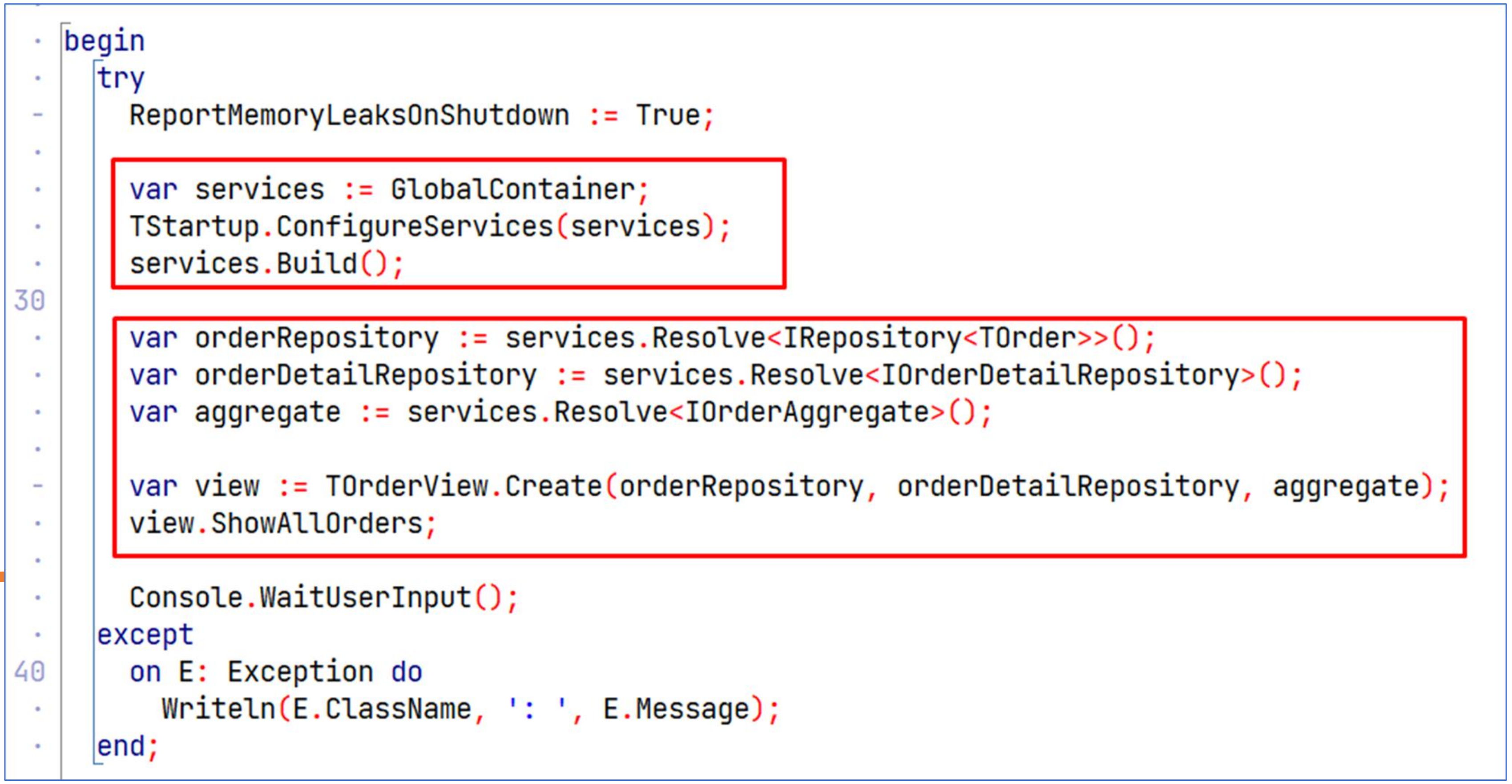


```
uses
  Spring.Persistence.Criteria.Properties;

type
  OrderDetail = class
  public
    class var OrderId: Prop;
    class var ProductId: Prop;
    class var UnitPrice: Prop;
    class var Quantity: Prop;
    class var Discount: Prop;

    class constructor Create;
  end;

implementation

{ OrderDetail }

class constructor OrderDetail.Create;
begin
  OrderId   := Prop.Create('OrderID');
  ProductId := Prop.Create('ProductID');
  UnitPrice := Prop.Create('UnitPrice');
  Quantity  := Prop.Create('Quantity');
  Discount  := Prop.Create('Discount');
end;
```



```
Where(OrderDetail.OrderId = orderId);
```

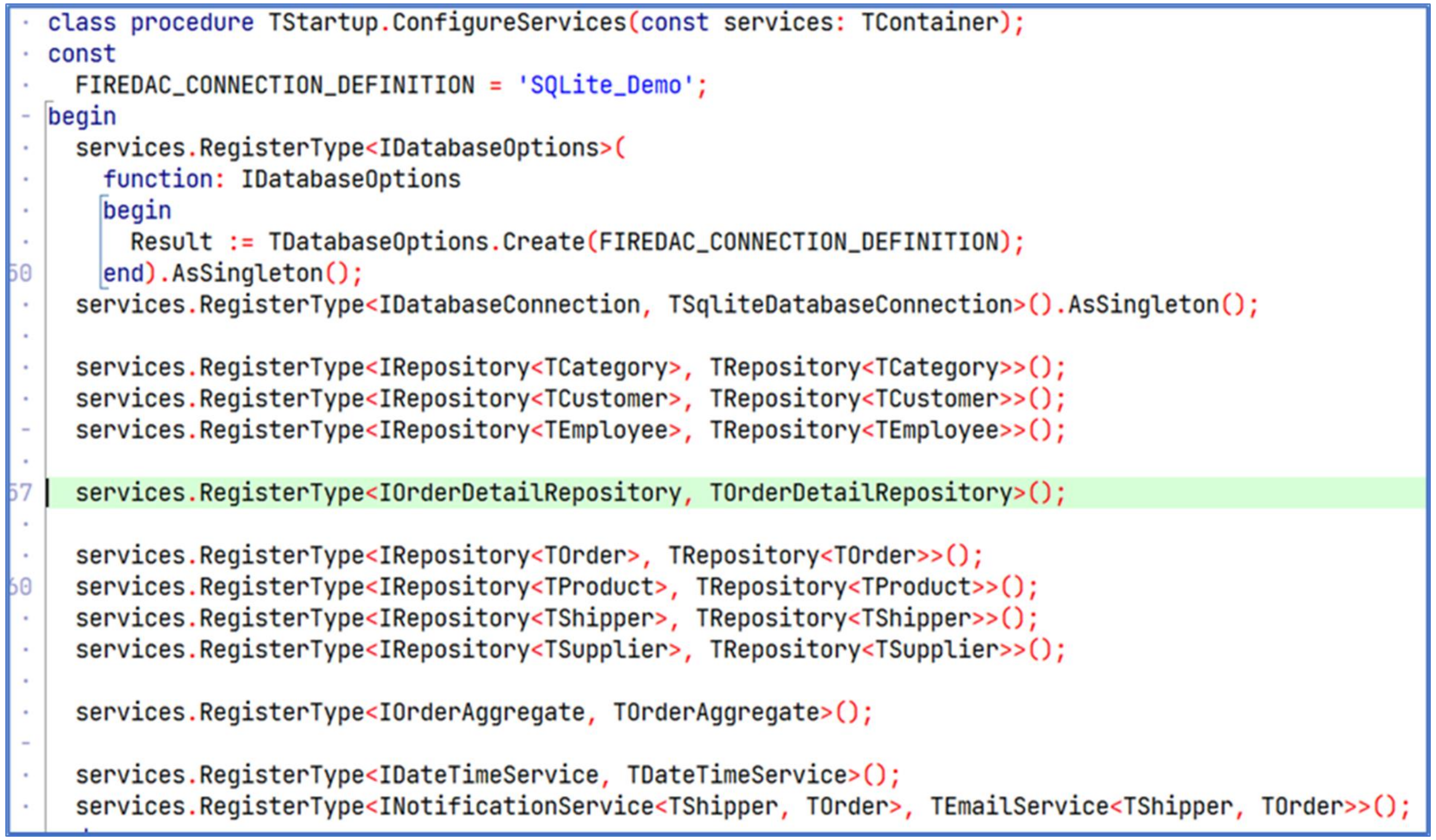
Application



```
begin
  try
    ReportMemoryLeaksOnShutdown := True;

    var services := GlobalContainer;
    TStartup.ConfigureServices(services);
    services.Build();

    var orderRepository := services.Resolve<IRepository<TOrder>>();
    var orderDetailRepository := services.Resolve<IOrderDetailRepository>();
    var aggregate := services.Resolve<IOrderAggregate>();

    var view := TOrderView.Create(orderRepository, orderDetailRepository, aggregate);
    view.ShowAllOrders;

    Console.WaitUserInput();
  except
    on E: Exception do
      Writeln(E.ClassName, ': ', E.Message);
  end;
```

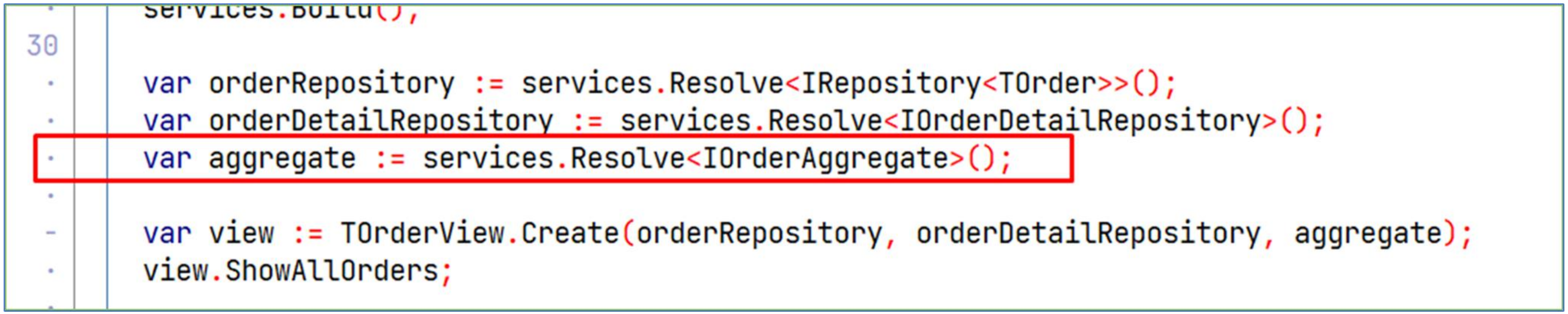
Classe “Startup”

- Configurações e Opções
- Certificados
- Connection Strings
- Injeção de Dependência
- Registrar Serviços
- Ciclo de Vida dos Serviços



```
class procedure TStartup.ConfigureServices(const services: TContainer);
const
  FIREDAC_CONNECTION_DEFINITION = 'SQLite_Demo';
begin
  services.RegisterType<IDatabaseOptions>(
    function: IDatabaseOptions
    begin
      Result := TDatabaseOptions.Create(FIREDAC_CONNECTION_DEFINITION);
    end).AsSingleton();
  services.RegisterType<IDatabaseConnection, TSqliteDatabaseConnection>().AsSingleton();

  services.RegisterType<IRepository<TCategory>, TRepository<TCategory>>();
  services.RegisterType<IRepository<TCustomer>, TRepository<TCustomer>>();
  services.RegisterType<IRepository<TEmployee>, TRepository<TEmployee>>();

  services.RegisterType<IOrderDetailRepository, TOrderDetailRepository>();

  services.RegisterType<IRepository<TOrder>, TRepository<TOrder>>();
  services.RegisterType<IRepository<TProduct>, TRepository<TProduct>>();
  services.RegisterType<IRepository<TShipper>, TRepository<TShipper>>();
  services.RegisterType<IRepository<TSupplier>, TRepository<TSupplier>>();

  services.RegisterType<IOrderAggregate, TOrderAggregate>();

  services.RegisterType<IDateTimeService, TDateTimeService>();
  services.RegisterType<INotificationService<TShipper, TOrder>, TEmailService<TShipper, TOrder>>();
```

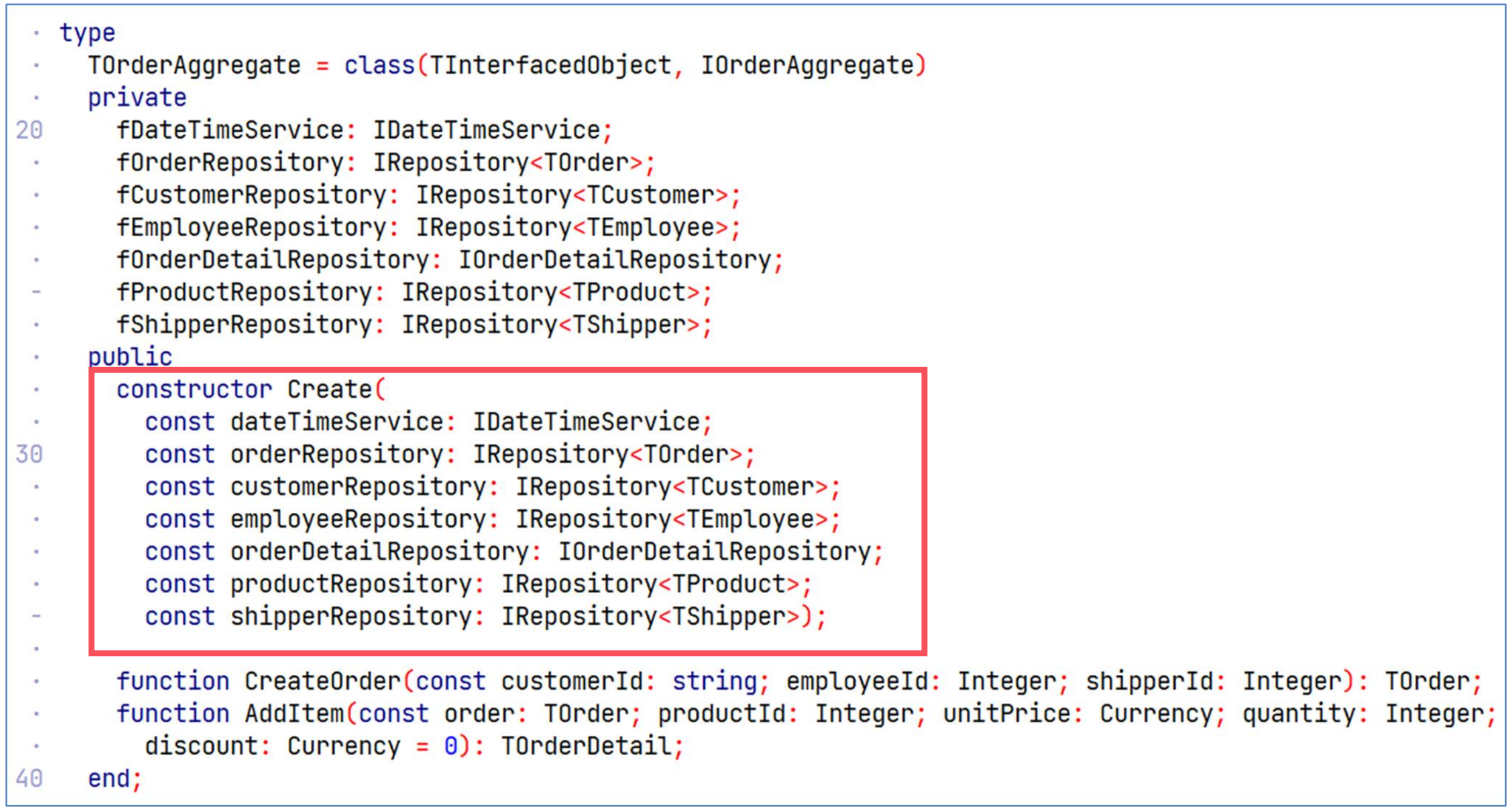
Injeção de Dependência



```
  services.Build();

  var orderRepository := services.Resolve<IRepository<TOrder>>();
  var orderDetailRepository := services.Resolve<IOrderDetailRepository>();
  var aggregate := services.Resolve<IOrderAggregate>();

  var view := TOrderView.Create(orderRepository, orderDetailRepository, aggregate);
  view.ShowAllOrders;
```

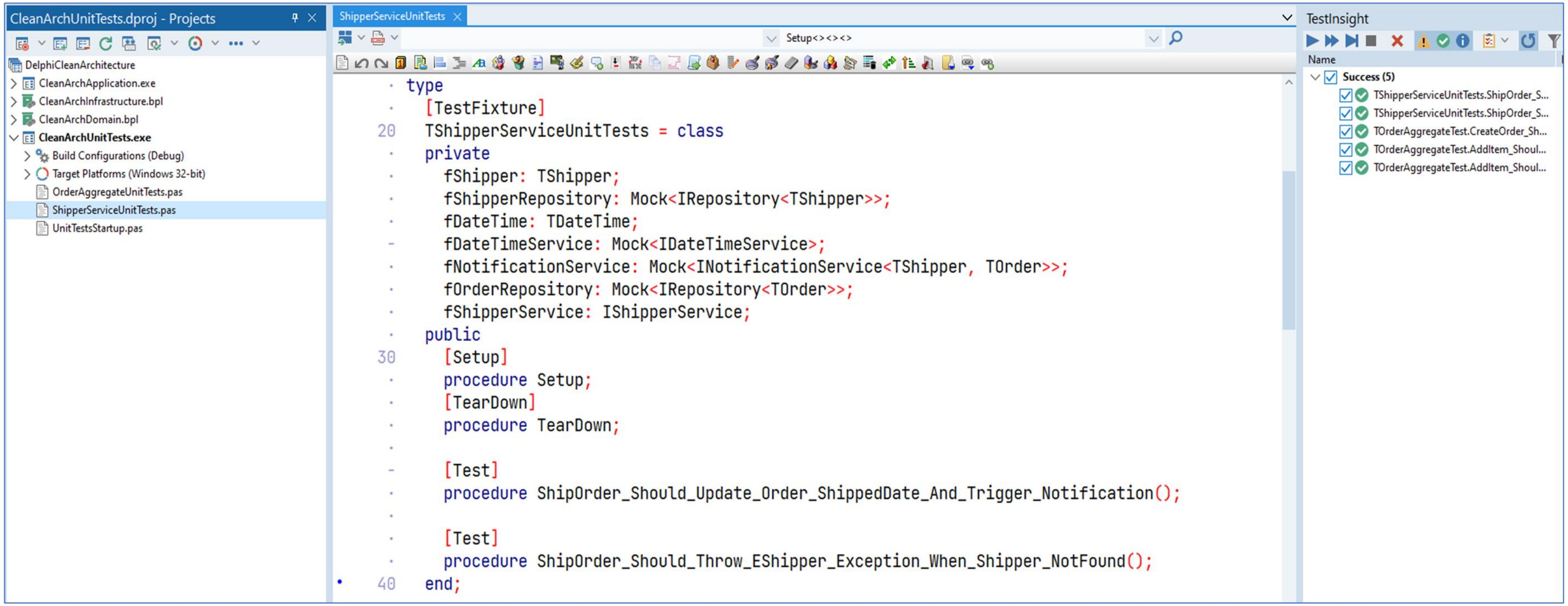
Injeção de Dependência

- Constructor



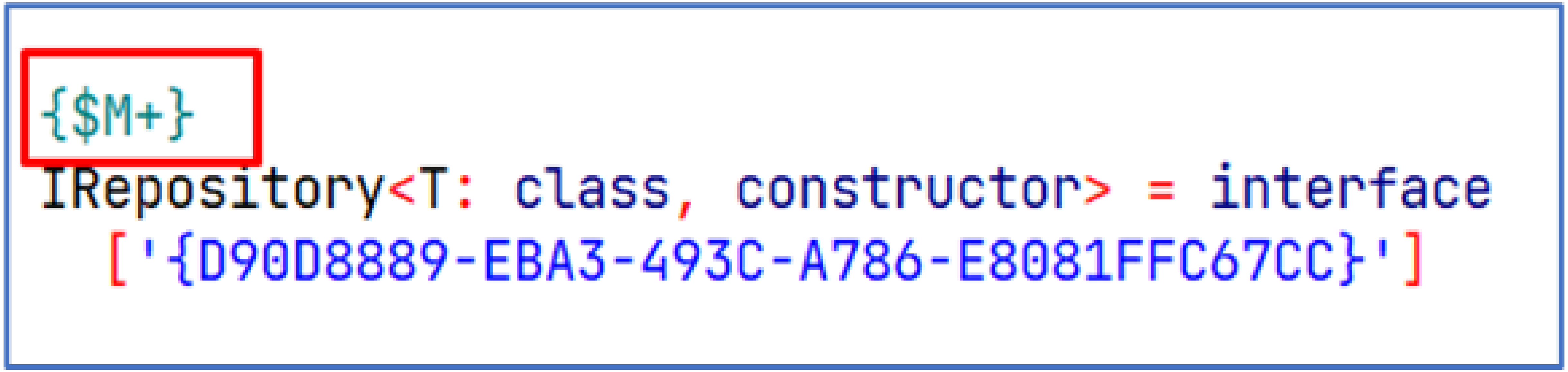
```
type
  TOrderAggregate = class(TInterfacedObject, IOrderAggregate)
  private
    fDateTimeService: IDateTimeService;
    fOrderRepository: IRepository<TOrder>;
    fCustomerRepository: IRepository<TCustomer>;
    fEmployeeRepository: IRepository<TEmployee>;
    fOrderDetailRepository: IOrderDetailRepository;
    fProductRepository: IRepository<TProduct>;
    fShipperRepository: IRepository<TShipper>;
  public
    constructor Create(
      const dateTimeService: IDateTimeService;
      const orderRepository: IRepository<TOrder>;
      const customerRepository: IRepository<TCustomer>;
      const employeeRepository: IRepository<TEmployee>;
      const orderDetailRepository: IOrderDetailRepository;
      const productRepository: IRepository<TProduct>;
      const shipperRepository: IRepository<TShipper>);

    function CreateOrder(const customerId: string; employeeId: Integer; shipperId: Integer): TOrder;
    function AddItem(const order: TOrder; productId: Integer; unitPrice: Currency; quantity: Integer;
      discount: Currency = 0): TOrderDetail;
  end;
```

Testes Unitários


CleanArchUnitTests.dproj - Projects
DelphiCleanArchitecture
CleanArchApplication.exe
CleanArchInfrastructure.bpl
CleanArchDomain.bpl
CleanArchUnitTests.exe
Build Configurations (Debug)
Target Platforms (Windows 32-bit)
OrderAggregateUnitTests.pas
ShipperServiceUnitTests.pas
UnitTestsStartup.pas
ShipperServiceUnitTests
Setup<><><>
TestInsight
Name
Success (5)
TShipperServiceUnitTests.ShipOrder_S...
TShipperServiceUnitTests.ShipOrder_S...
TOrderAggregateTest.CreateOrder_Sh...
TOrderAggregateTest.AddItem_Shoul...
TOrderAggregateTest.AddItem_Shoul...

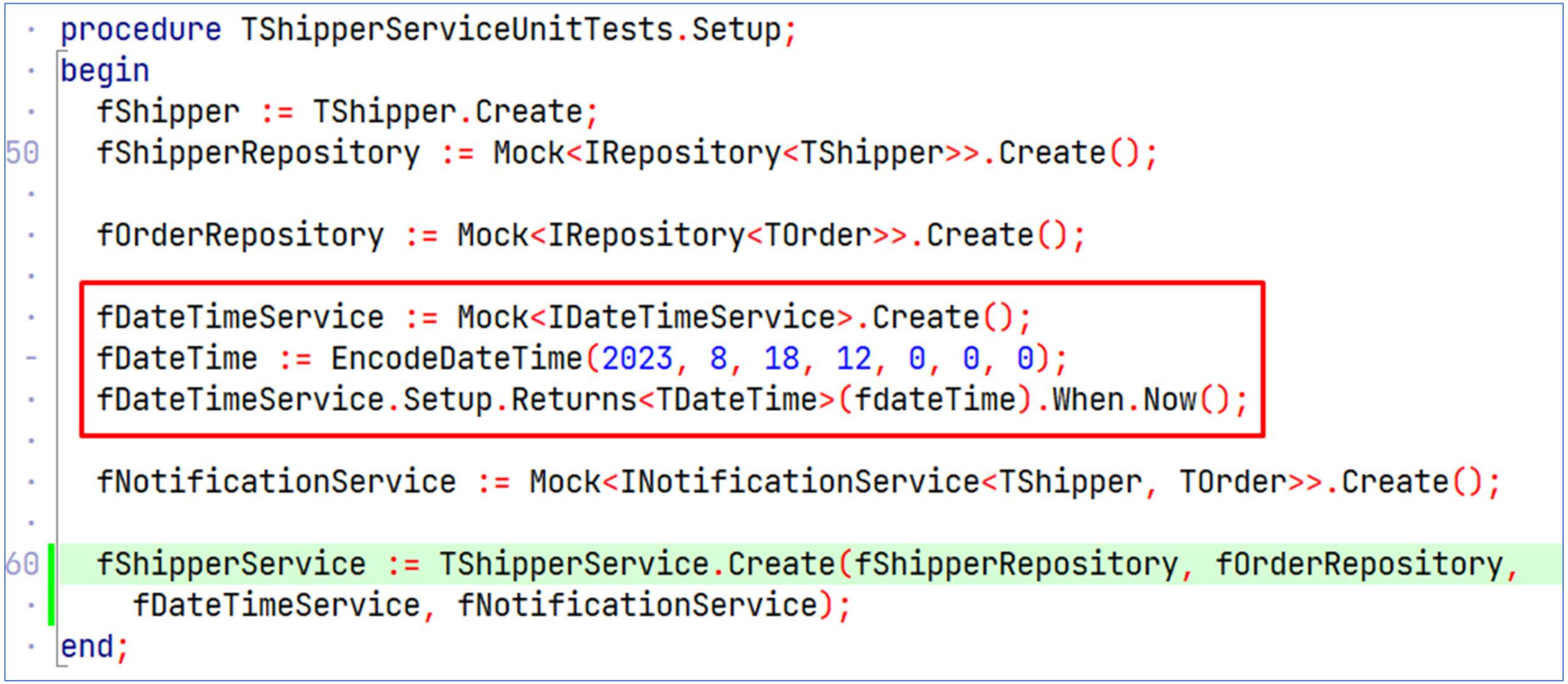
```
type
  [TestFixture]
  TShipperServiceUnitTests = class
  private
    fShipper: TShipper;
    fShipperRepository: Mock<IRepository<TShipper>>;
    fDateTime: TDateTime;
    fDateTimeService: Mock<IDateTimeService>;
    fNotificationService: Mock<INotificationService<TShipper, TOrder>>;
    fOrderRepository: Mock<IRepository<TOrder>>;
    fShipperService: IShipperService;
  public
    [Setup]
    procedure Setup;
    [TearDown]
    procedure TearDown;

    [Test]
    procedure ShipOrder_Should_Update_Order_ShippedDate_And_Trigger_Notification();

    [Test]
    procedure ShipOrder_Should_Throw_EShipper_Exception_When_Shipper_NotFound();
  end;
```

Ativar RTTI para Mocking



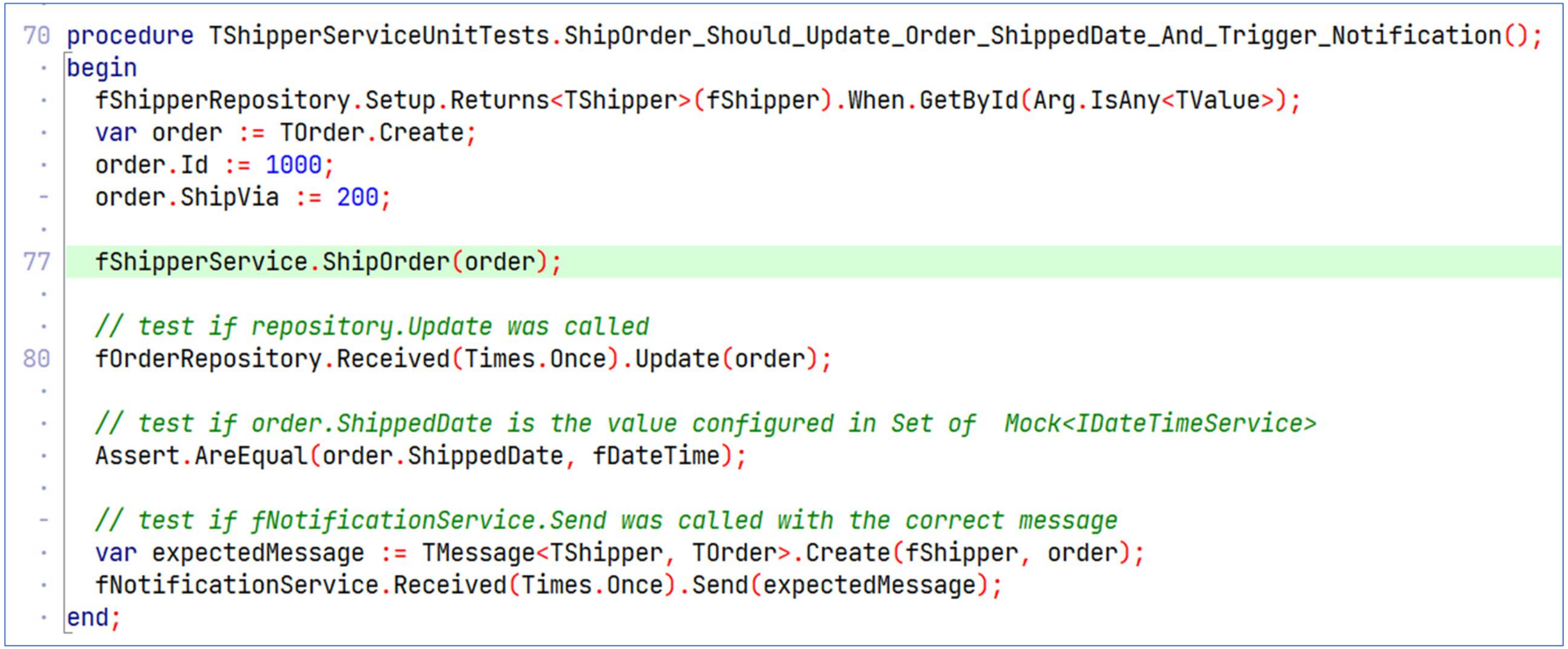
```
{$M+}
IRepository<T: class, constructor> = interface
  ['{D90D8889-EBA3-493C-A786-E8081FFC67CC}']
```

Mocking Setup



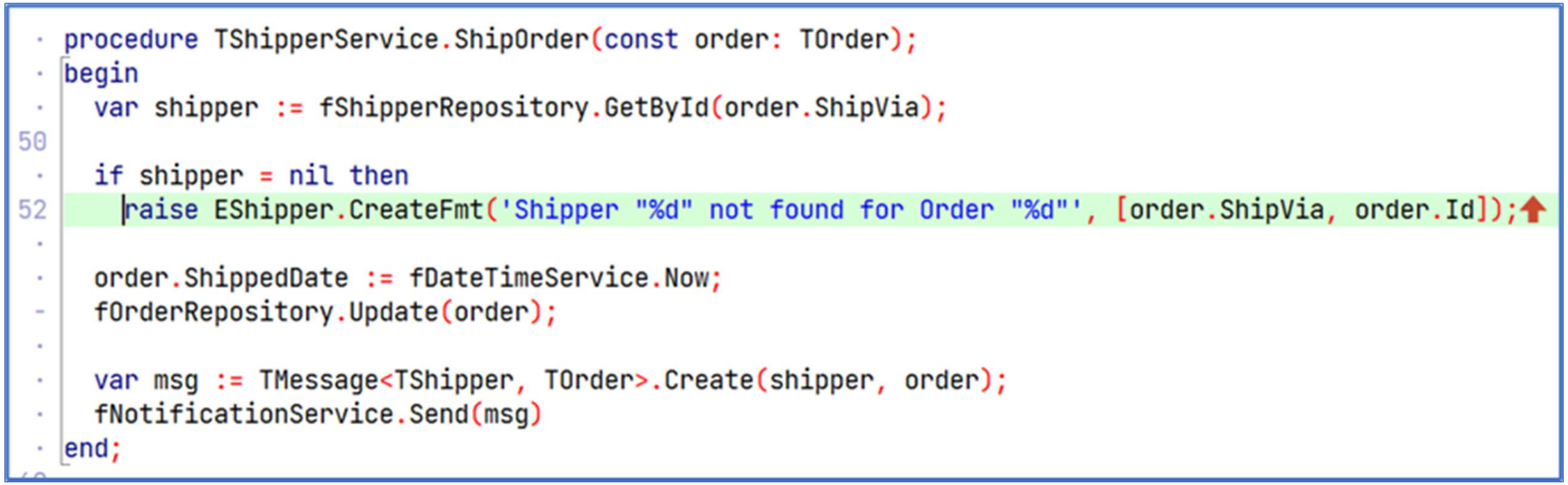
```
procedure TShipperServiceUnitTests.Setup;
begin
  fShipper := TShipper.Create;
  fShipperRepository := Mock<IRepository<TShipper>>.Create();

  fOrderRepository := Mock<IRepository<TOrder>>.Create();

  fDateTimeService := Mock<IDateTimeService>.Create();
  fDateTime := EncodeDateTime(2023, 8, 18, 12, 0, 0, 0);
  fDateTimeService.Setup.Returns<TDateTime>(fdateTime).When.Now();

  fNotificationService := Mock<INotificationService<TShipper, TOrder>>.Create();

  fShipperService := TShipperService.Create(fShipperRepository, fOrderRepository,
    fDateTimeService, fNotificationService);
end;
```

Test Case



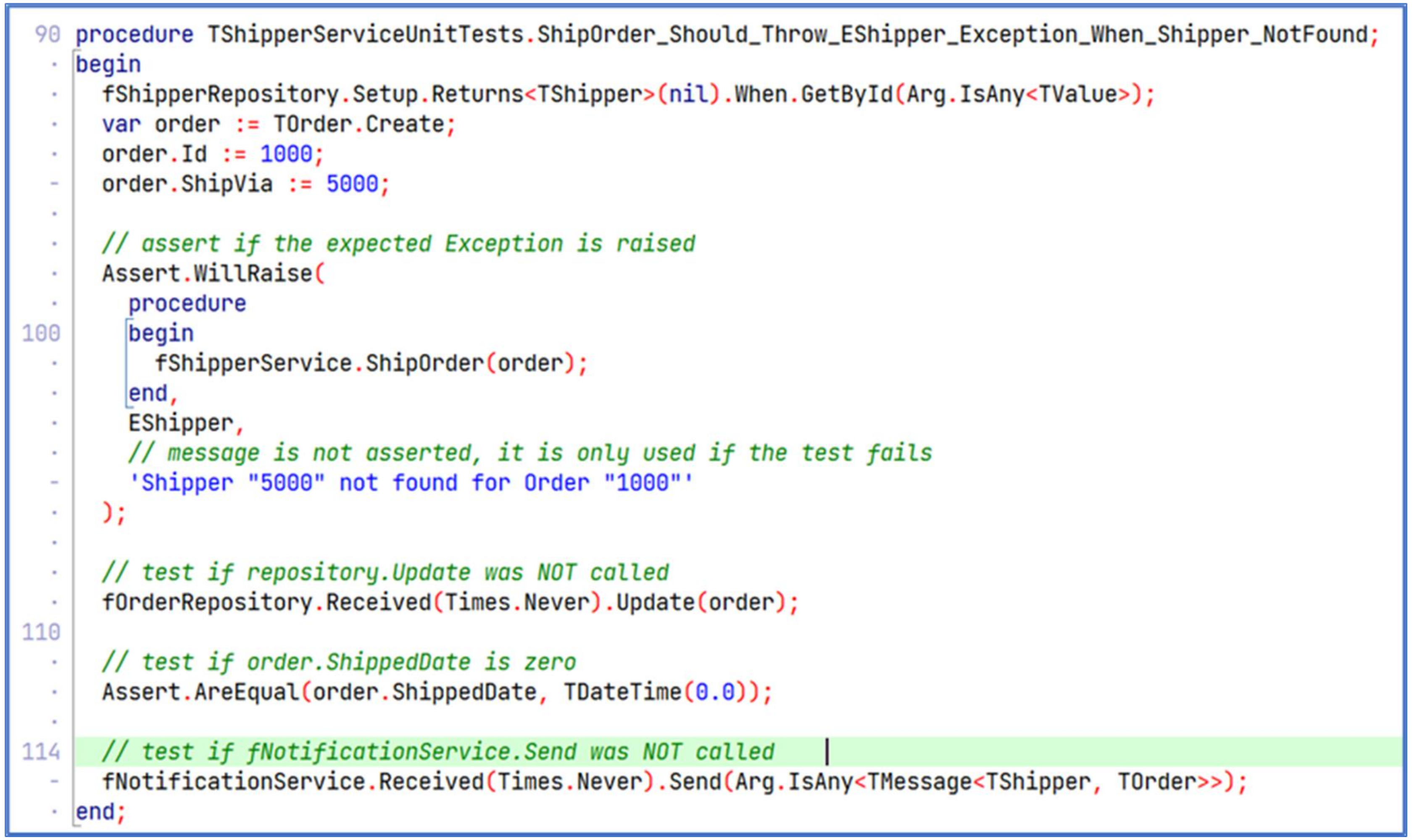
```
procedure TShipperServiceUnitTests.ShipOrder_Should_Update_Order_ShippedDate_And_Trigger_Notification();
begin
  fShipperRepository.Setup.Returns<TShipper>(fShipper).When.GetById(Arg.IsAny<TValue>);
  var order := TOrder.Create;
  order.Id := 1000;
  order.ShipVia := 200;

  fShipperService.ShipOrder(order);

  // test if repository.Update was called
  fOrderRepository.Received(Times.Once).Update(order);

  // test if order.ShippedDate is the value configured in Set of  Mock<IDateTimeService>
  Assert.AreEqual(order.ShippedDate, fDateTime);

  // test if fNotificationService.Send was called with the correct message
  var expectedMessage := TMessage<TShipper, TOrder>.Create(fShipper, order);
  fNotificationService.Received(Times.Once).Send(expectedMessage);
end;
```

Code Coverage



```
procedure TShipperService.ShipOrder(const order: TOrder);
begin
  var shipper := fShipperRepository.GetById(order.ShipVia);

  if shipper = nil then
    raise EShipper.CreateFmt('Shipper "%d" not found for Order "%d"', [order.ShipVia, order.Id]);

  order.ShippedDate := fDateTimeService.Now;
  fOrderRepository.Update(order);

  var msg := TMessage<TShipper, TOrder>.Create(shipper, order);
  fNotificationService.Send(msg)
end;
```

Teste de Fluxo Alternativo - Exceções



```
procedure TShipperServiceUnitTests.ShipOrder_Should_Throw_EShipper_Exception_When_Shipper_NotFound;
begin
  fShipperRepository.Setup.Returns<TShipper>(nil).When.GetById(Arg.IsAny<TValue>);
  var order := TOrder.Create;
  order.Id := 1000;
  order.ShipVia := 5000;

  // assert if the expected Exception is raised
  Assert.WillRaise(
    procedure
    begin
      fShipperService.ShipOrder(order);
    end,
    EShipper,
    // message is not asserted, it is only used if the test fails
    'Shipper "5000" not found for Order "1000"'
  );

  // test if repository.Update was NOT called
  fOrderRepository.Received(Times.Never).Update(order);

  // test if order.ShippedDate is zero
  Assert.AreEqual(order.ShippedDate, TDateTime(0.0));

  // test if fNotificationService.Send was NOT called
  fNotificationService.Received(Times.Never).Send(Arg.IsAny<TMessage<TShipper, TOrder>>);
end;
```

Obrigado

O uso de padrões de projetos e testes unitários é mais que apenas uma técnica ou uma ferramenta.

É uma forma de pensar e criar sistemas eficientes, confiáveis e sustentáveis.

É um meio de expresser sua criatividade, habilidade e paixão pela programação

"The only way to go fast is to go well."

Uncle Bob

Contato

- Cesar Romero
- @cesarliws
- cesarliws@gmail.com
- https://www.linkedin.com/in/cesarliws
- https://github.com/cesarliws

Obrigado

Avaliação - **O que achou da palestra?**

Acesse o link do QR Code e responda a pesquisa:

- ○ Best Practices
- ○ 16:30 | Como usar padrões de projeto e testes unitários para criar sistemas de alta qualidade | Cesar Romero

Contato

- Cesar Romero
- @cesarliws
- cesarliws@gmail.com
- https://www.linkedin.com/in/cesarliws
- https://github.com/cesarliws