...aPaulita.UsesCase\Specifications\OrderHeaderSpecification\AddressSpecification.cs 1

```
1  namespace LaPaulita.UsesCase.Specifications
2  {
3      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
4      {
5          private partial void IsAddressValid()
6          {
7              if (string.IsNullOrEmpty(entity.ShippingAddress))
8              {
9                  validationErrors.Add(new ValidationErrorDto
10                 {
11                     PropertyName = "ShippingAddress",
12                     ErrorMessage = "La dirección de envío es requerida."
13                 });
14             }
15
16             if (entity.ShippingAddress.Length > 50)
17             {
18                 validationErrors.Add(new ValidationErrorDto
19                 {
20                     PropertyName = "ShippingAddres",
21                     ErrorMessage = "La dirección de envío no puede exceder los 50 caracteres."
22                 });
23             }
24         }
25     }
26 }
27
```

...a\LaPaulita.UsesCase\Specifications\OrderHeaderSpecification\CitySpecification.cs 1

```
1  using LaPaulita.Sales.BusinessRules.DTOs;
2  using LaPaulita.Sales.BusinessRules.Interface;
3
4  namespace LaPaulita.UsesCase.Specifications
5  {
6      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
7      {
8          private partial void IsCityValid()
9          {
10             if (entity.ShippingCity <= 0)
11             {
12                 validationErrors.Add(new ValidationErrorDto
13                 {
14                     PropertyName = "SippingCity",
15                     ErrorMessage = "La ciudad de entrega es obligatorio."
16                 });
17             }
18         }
19     }
20 }
21
```

...Paulita.UsesCase\Specifications\OrderHeaderSpecification\ClientIdSpecification.cs 1

```
1  namespace LaPaulita.UsesCase.Specifications
2  {
3      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
4      {
5          private partial void IsClientIdValid()
6          {
7              if (entity.ClientId <= 0)
8              {
9                  validationErrors.Add(new ValidationErrorDto
10                 {
11                     PropertyName = "ClientId",
12                     ErrorMessage = "El Id del cliente es obligatorio."
13                 });
14             }
15         }
16     }
17 }
18
```

...LaPaulita.UsesCase\Specifications\OrderHeaderSpecification\CountrSpecification.cs 1

```
1  namespace LaPaulita.UsesCase.Specifications
2  {
3      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
4      {
5          private partial void IsCountryValid()
6          {
7              if (entity.ShippingCountry <= 0)
8              {
9                  validationErrors.Add(new ValidationErrorDto
10                 {
11                     PropertyName = "SippingCountry",
12                     ErrorMessage = "El país de entrega es obligatorio."
13                 });
14             }
15         }
16     }
17 }
18
```

...ia\LaPaulita.UsesCase\Specifications\OrderHeaderSpecification\ZipSpecification.cs 1

```
1  using System.Text.RegularExpressions;
2
3  namespace LaPaulita.UsesCase.Specifications
4  {
5      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
6      {
7          private partial void IsZipValid()
8          {
9              if (string.IsNullOrEmpty(entity.ShippingZip))
10             {
11                 validationErrors.Add(new ValidationErrorDto
12                 {
13                     PropertyName = "ShippingZip",
14                     ErrorMessage = "El código postal de entrega es obligatorio."
15                 });
16             }
17
18             if (entity.ShippingZip.Length != 4)
19             {
20                 validationErrors.Add(new ValidationErrorDto
21                 {
22                     PropertyName = "ShippingZip",
23                     ErrorMessage = "El código postal de entrega debe contener 4 caracteres."
24                 });
25             }
26
27             // Patrón de expresión regular para verificar si solo contiene números
28             string pattern = @"^[0-9]+$";
29
30             // Verificar si la entrada coincide con el patrón
31             bool isMatch = Regex.IsMatch(entity.ShippingZip, pattern);
32
33             if (!isMatch)
34             {
35                 validationErrors.Add(new ValidationErrorDto
36                 {
37                     PropertyName = "SippingZip",
38                     ErrorMessage = "El código postal de entrega debe contener solo números."
39                 });
40             }
41         }
42     }
43 }
44
```

...3\LaPaulita - copia\LaPaulita.UsesCase\Specifications\OrderHeaderSpecification.cs 1

```
1  namespace LaPaulita.UsesCase.Specifications
2  {
3      public partial class OrderHeaderSpecification : ISpecification<OrderHeaderDto>
4      {
5          readonly List<ValidationErrorDto> validationErrors = new List<ValidationErrorDto>();
6          readonly OrderHeaderDto entity;
7
8          public OrderHeaderSpecification(OrderHeaderDto entity)
9          {
10             this.entity = entity;
11         }
12
13         public List<ValidationErrorDto> IsValid()
14         {
15             IsClientIdValid();
16             IsAddressValid();
17             IsCityValid();
18             IsCountryValid();
19             IsZipValid();
20
21             return validationErrors;
22         }
23
24         private partial void IsClientIdValid();
25         private partial void IsAddressValid();
26         private partial void IsCityValid();
27         private partial void IsCountryValid();
28         private partial void IsZipValid();
29     }
30 }
31
```

...maciónIes2023\LaPaulita - copia\LaPaulita.UsesCase\Create\CreateOrderIteractor.cs 1

```
1  using LaPaulita.Sales.BusinessRules.Agregates;
2  using LaPaulita.Sales.BusinessRules.Interface.Presenters;
3  using LaPaulita.Sales.BusinessRules.Interface.Repositories;
4  using LaPaulita.Sales.BusinessRules.Wrappers;
5  using LaPaulita.UsesCase.Specifications;
6
7  namespace LaPaulita.Sales.UsesCase.Create
8  {
9      /// <summary>
10     /// <b>Use Case Interactor</b>. Es el elemento que contiene el código con la lógica de
11     /// negocios que resuelve un caso de uso. Este elemento implementa la abstracción
12     /// representada por el elemento <i>Use Case Input Port</i>. En términos de programación
13     /// orientada a objetos, el <b>Interactor</b> es una clase que implementa una
14     /// Interface o clase abstracta <i>(InputPort)</i>.
15     /// </summary>
16     public class CreateOrderIteractor : ICreateOrderInputPort
17     {
18         //readonly ICreateOrderOutputPort _outputPort;
19         readonly ISalesCommandRepository _repository;
20         readonly ICreateOrderPresenter _presenter;
21
22         public CreateOrderIteractor(ICreateOrderOutputPort outputPort, ISalesCommandRepository repository,
               ICreateOrderPresenter presenter)
23         {
24             //_outputPort = outputPort;
25             _repository = repository;
26             _presenter = presenter;
27         }
28
29         public async Task Handle(OrderHeaderDto createOrderDto)
30         {
31             // Instanciamos un objeto del tipo List<ValidationErrorDto> y le asignamos
32             // lo que nos devuelva el método privado ValidateOrder.
33
34             List<ValidationErrorDto> validationErrors = new List<ValidationErrorDto>();
35             validationErrors = ValidateOrder(createOrderDto);
36             WrappersSalesOrder order = new();
37             // Consultamos si la lista validationErrors posee algún elemento.
38             if (validationErrors.Count > 0)
39             {
```

...mación Ies2023\LaPaulita - copia\LaPaulita.UsesCase\Create\CreateOrderIteractor.cs 2

```
                // Si la lista poseía algún elemento, es que hay por lo menos un error
                // Entonses retormaos el OutputPort al presentador con la lista de errores.

                order.Errors = validationErrors;
                await _presenter.Handle(order);
                return;
            }

            // Si no hay errores contiunuamos con la ejecución del código y
            // creamos la orden, luego guardamos los cambios y finalmente retornamos
            // el Id del registro creado.

            CreateOrder createOrder = CreateOrder.From(createOrderDto);
            try
            {
                await _repository.CreateOrder(createOrder);
                await _repository.SaveChange();
                order.OrderId = createOrder.Id;
                await _presenter.Handle(order);
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }

        }
        /// <summary>
        /// Método que valida los datos de la Orden de compra antes de ser presistida
        /// en la base de datos.
        /// <br/> <br/>
        /// <i>Posteriormente deberemos realizar la validación de la existencia de los
        /// datos que debene existir previamente a la orden de compra, como el id del
        /// cliente o el id del producto.</i>
        /// </summary>
        /// <param name="createOrderDto">Objeto que contiene las propiedades a validar</param>
        /// <returns>La lista de propiedades que no cumplen con la validación
        /// y la descripción del error específico.</returns>
        private List<ValidationErrorDto> ValidateOrder(OrderHeaderDto createOrderDto)
        {
            var specification = new OrderHeaderSpecification(createOrderDto);

            return specification.IsValid();
        }
    }
}

```

...e\ProgramaciónIes2023\LaPaulita - copia\LaPaulita.UsesCase\DependencyContainer.cs 1

```
namespace LaPaulita.Sales.UsesCase
{
    public static class DependencyContainer
    {
        public static IServiceCollection AddServicesUseCase(this IServiceCollection services)
        {
            services.AddScoped<ICreateOrderInputPort, CreateOrderIteractor>();

            return services;
        }
    }
}

```