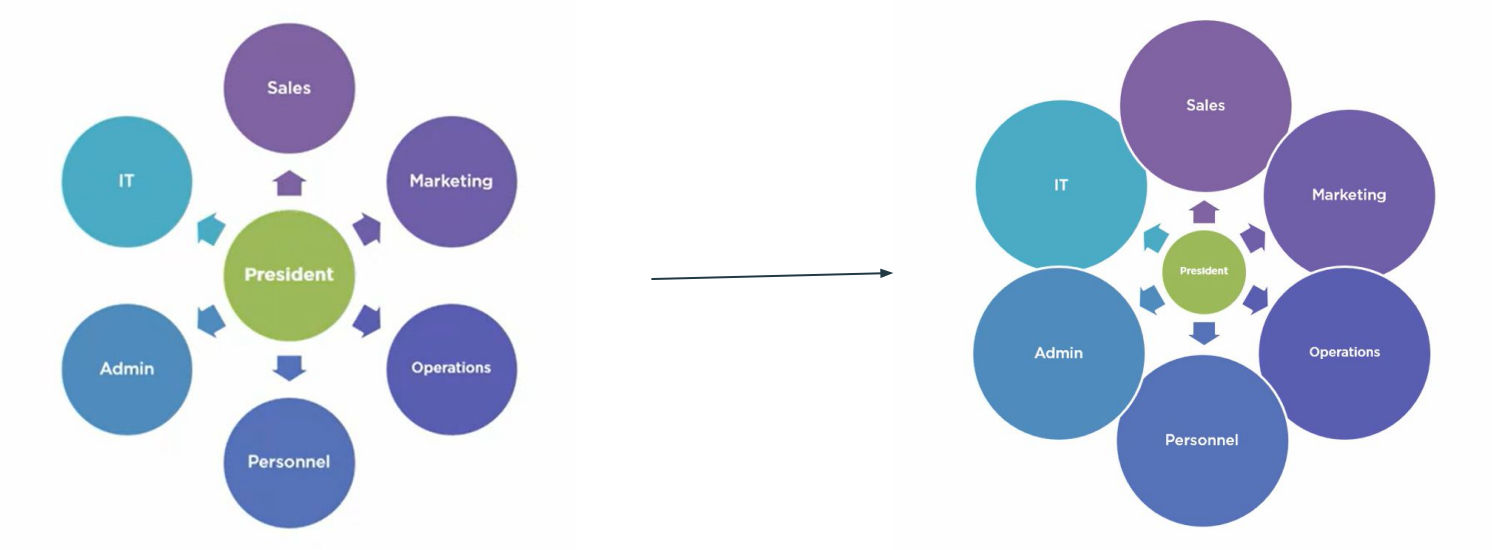
C#: Separando Responsabilidades, Relaciones y Reusabilidad

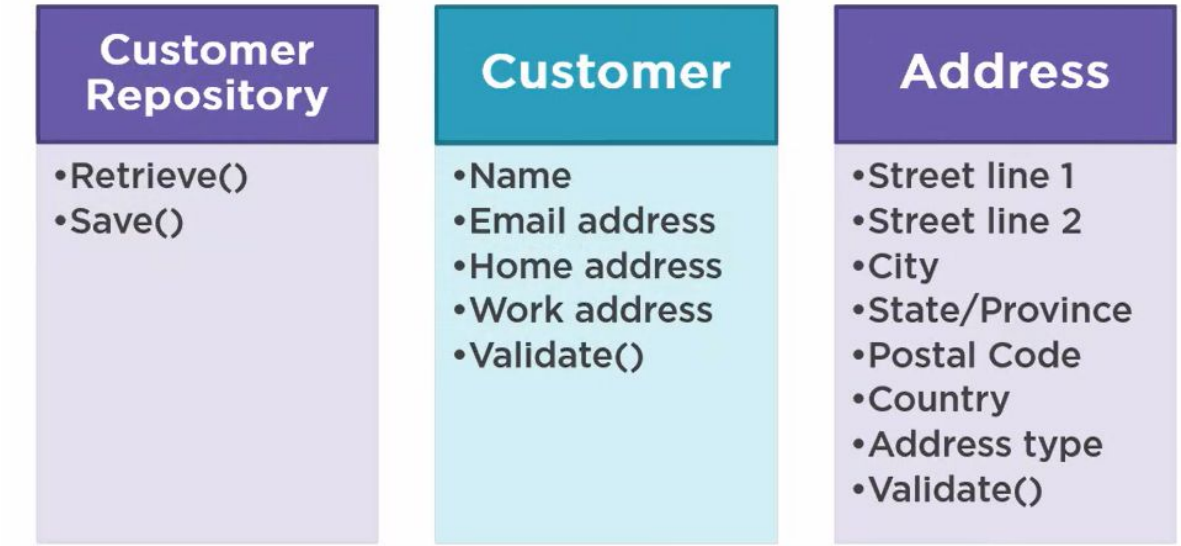
Responsabilidades




Name
Email Address
Postal Address
Validate
Retrieve
Save
Customer

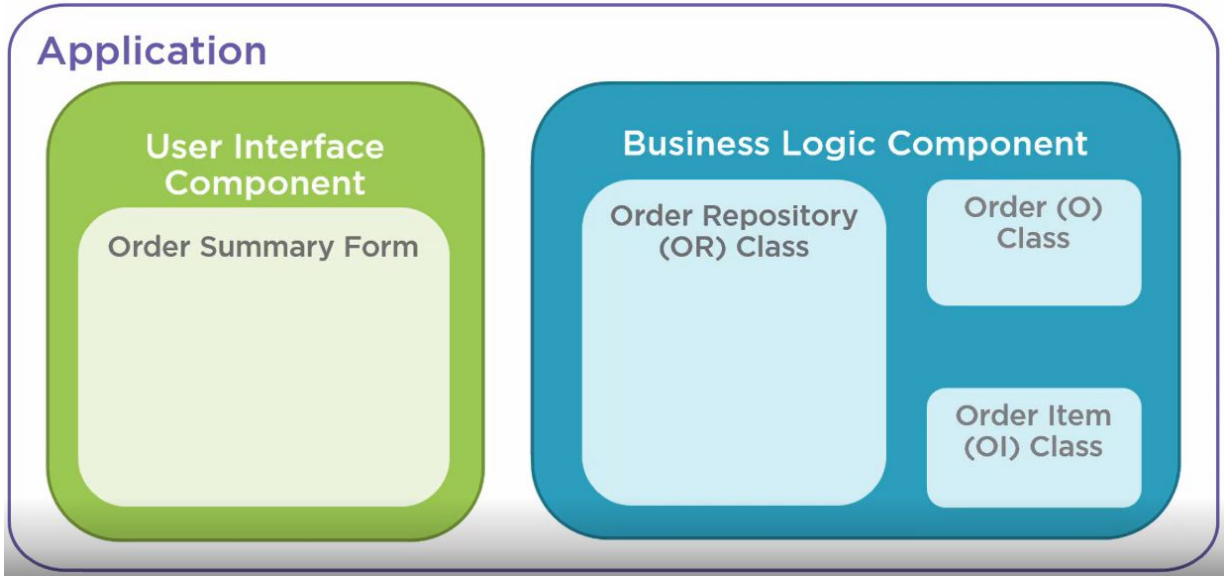
Alta Cohesión y Bajo Acoplamiento

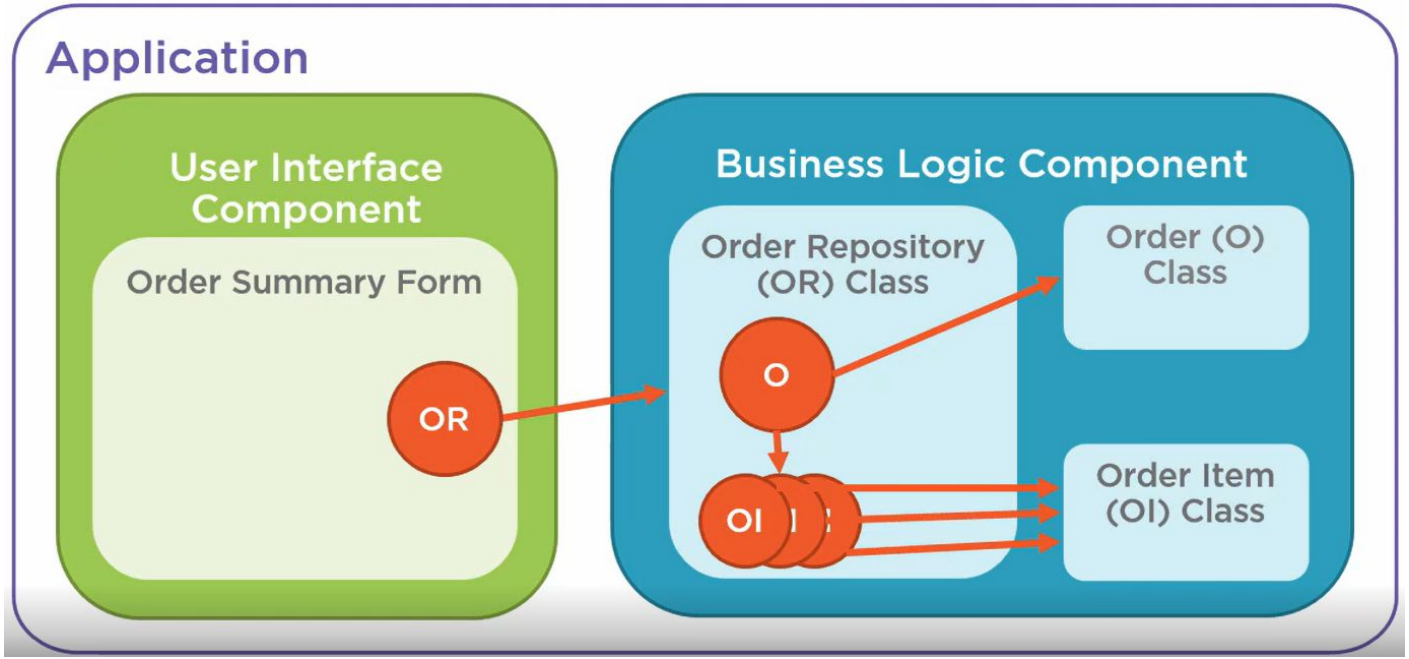
Separando Responsabilidades

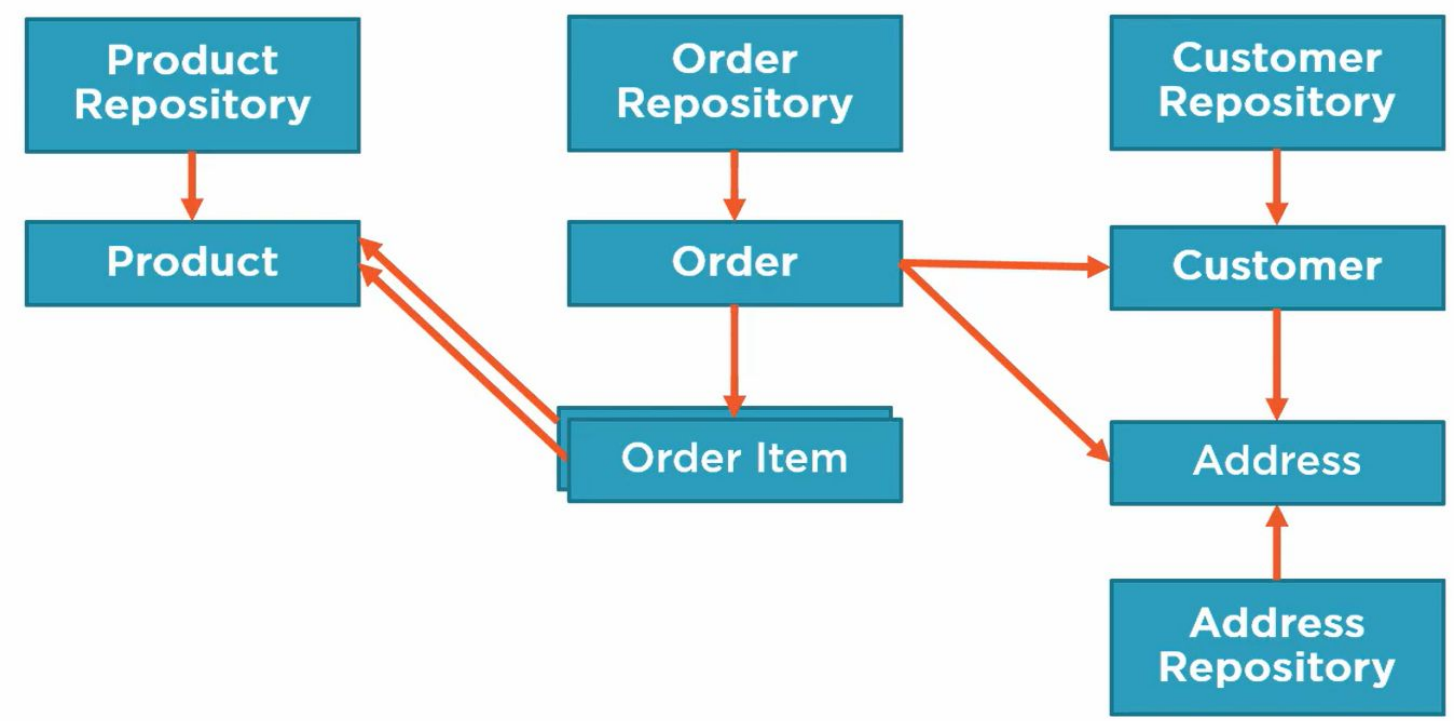

Customer Repository
•Retrieve()
•Save()
Customer
•Name
•Email address
•Home address
•Work address
•Validate()
Address
•Street line 1
•Street line 2
•City
•State/Province
•Postal Code
•Country
•Address type
•Validate()


Customer
•Name
•Email address
•Home address
•Work address
•Validate()
Product
•Product name
•Description
•Current price
•Validate()
Order
•Customer
•Order date
•Shipping addr.
•Order items
•Validate()
Customer Repository
•Retrieve()
•Save()
Product Repository
•Retrieve()
•Save()
Order Repository
•Retrieve()
•Save()
Address
•Street line 1 + 2
•City
•State/Province
•Postal Code
•Country
•Address type
•Validate()

Estableciendo Relaciones




Application
User Interface Component
Order Summary Form
OR
Business Logic Component
Order Repository (OR) Class
O
OI
Order (O) Class
Order Item (OI) Class


Product Repository
Order Repository
Customer Repository
Product
Order
Customer
Order Item
Address
Address Repository

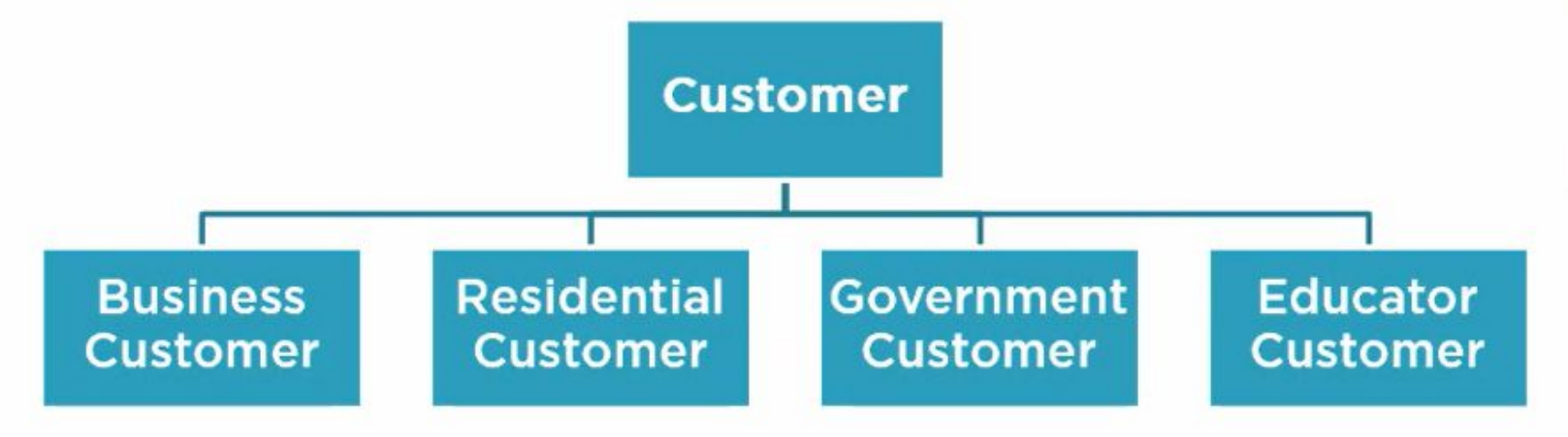
Tipos de Relaciones

- Colaboración (usa)
- Composición (tiene) (agregación / composición)
- Herencia

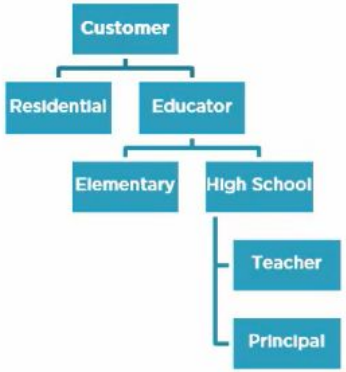
Herencia

"The new system must manage business, residential, government, and educator types of customers."

From the requirements


Customer
Business Customer
Residential Customer
Government Customer
Educator Customer

A class can only have one parent class



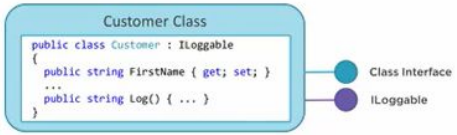
There can be any number of inheritance levels

Aprovechando la reusabilidad

Collaboration / Composition

Inheritance

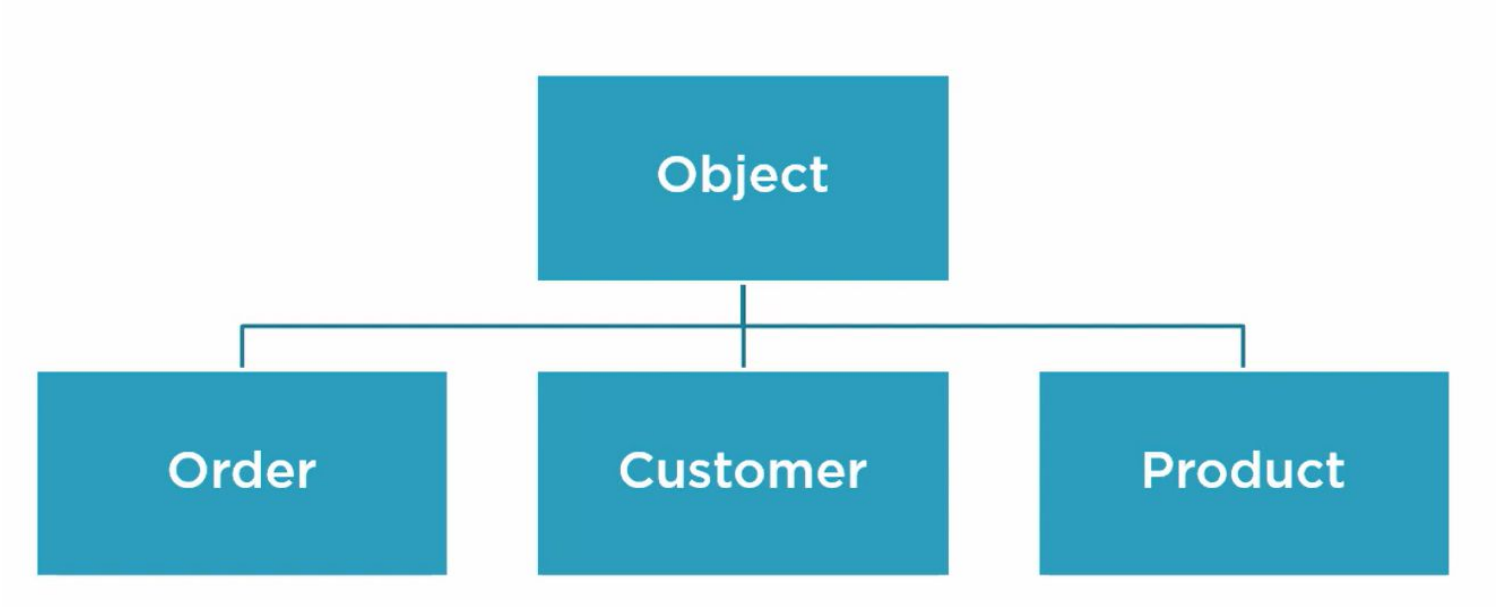
Components



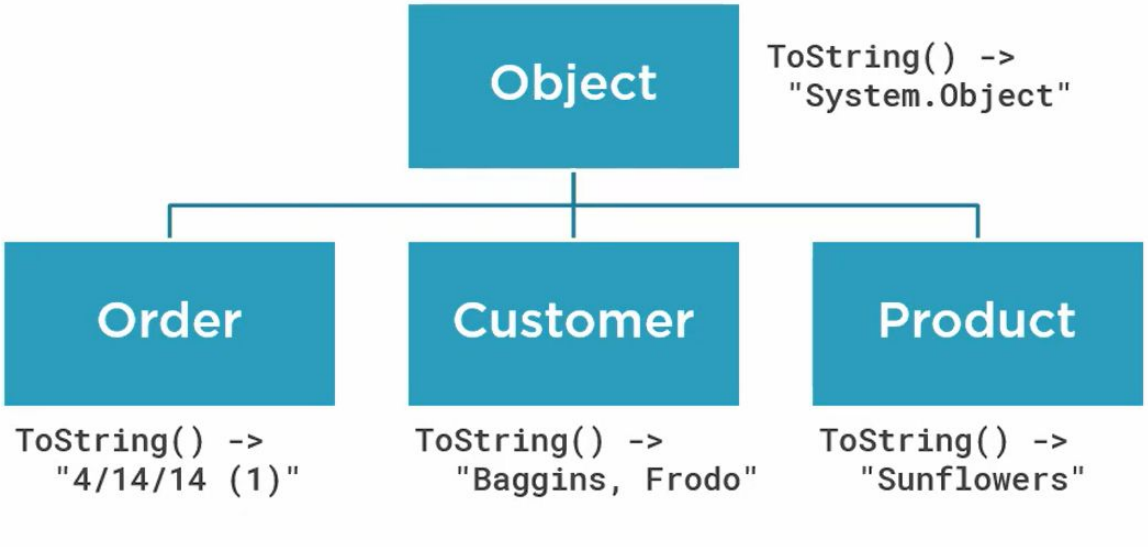
Interfaces

Copy/Paste

La clase Object



Polimorfismo



Construyendo una clase base

```
public bool IsNew { get; private set; }
```

```
public bool HasChanges { get; set; }
```

```
public bool IsValid => Validate();
```

```
public EntityStateOption EntityState { get; set; }
```

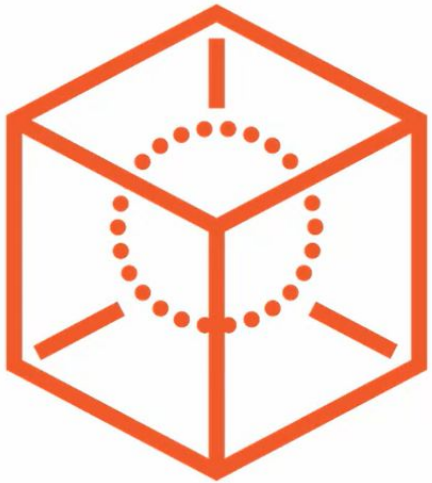
Clases abstractas / Clases concretas

Abstract Class	Concrete Class
Incomplete, with at least one property or method not implemented	Normal class
Cannot be instantiated	Can be instantiated
Intended for use as a base class	Can be used as a base class

```
public abstract class EntityBase
{

}
```

```
public class EntityBase
{

}
```

Clases Selladas (sealed)

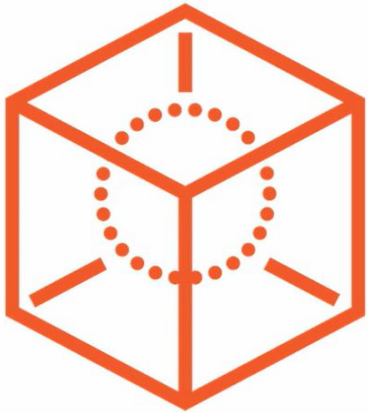


Cannot be extended through inheritance

Sealed using the `sealed` keyword

```
public sealed class Customer
{

}
```

Miembros Sellados



By default, class members are sealed and cannot be overridden

Expose members using

- Abstract
- Virtual

Abstract

Method signature as place holder with no implementation

Only use in abstract classes

Must be overridden by derived class

```
public abstract bool Validate();
```

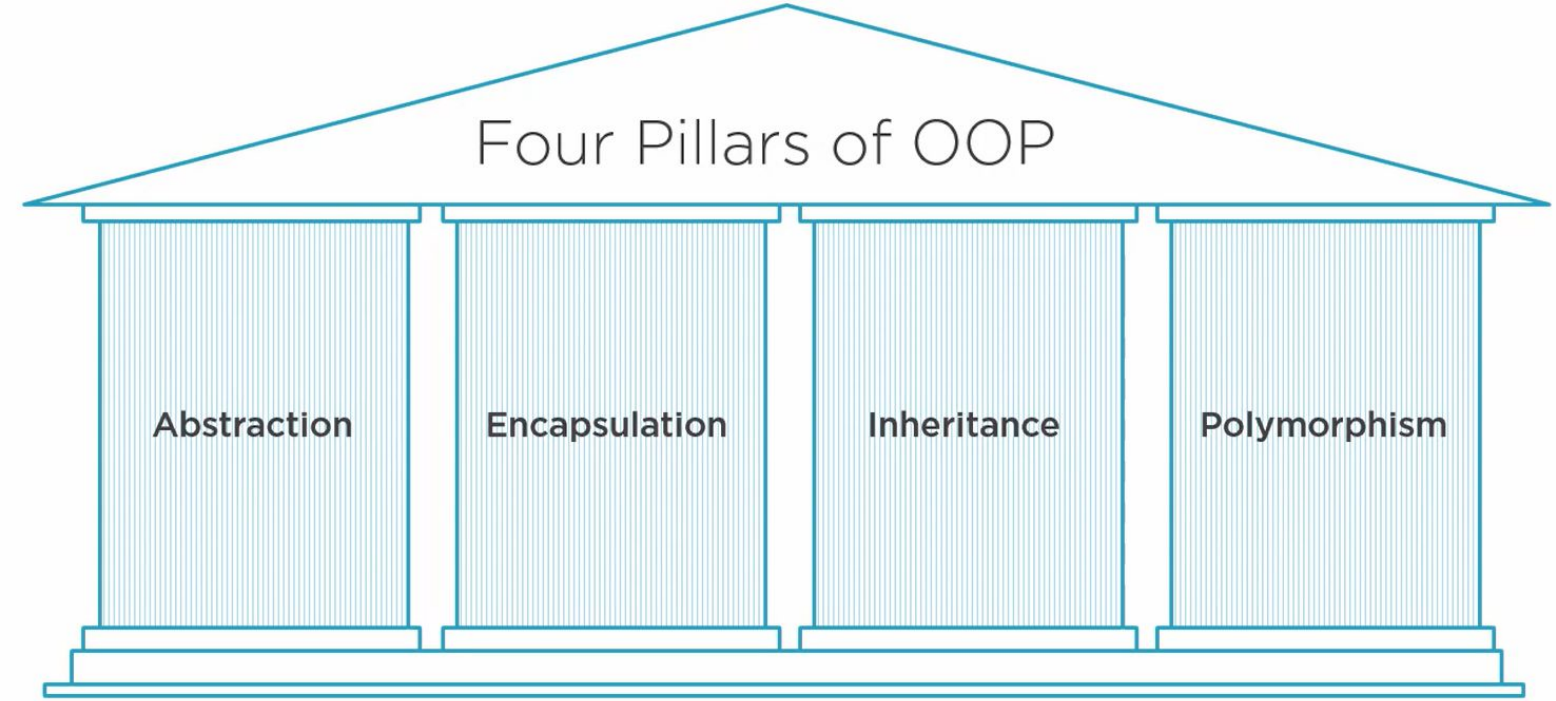
Virtual

Method with default implementation

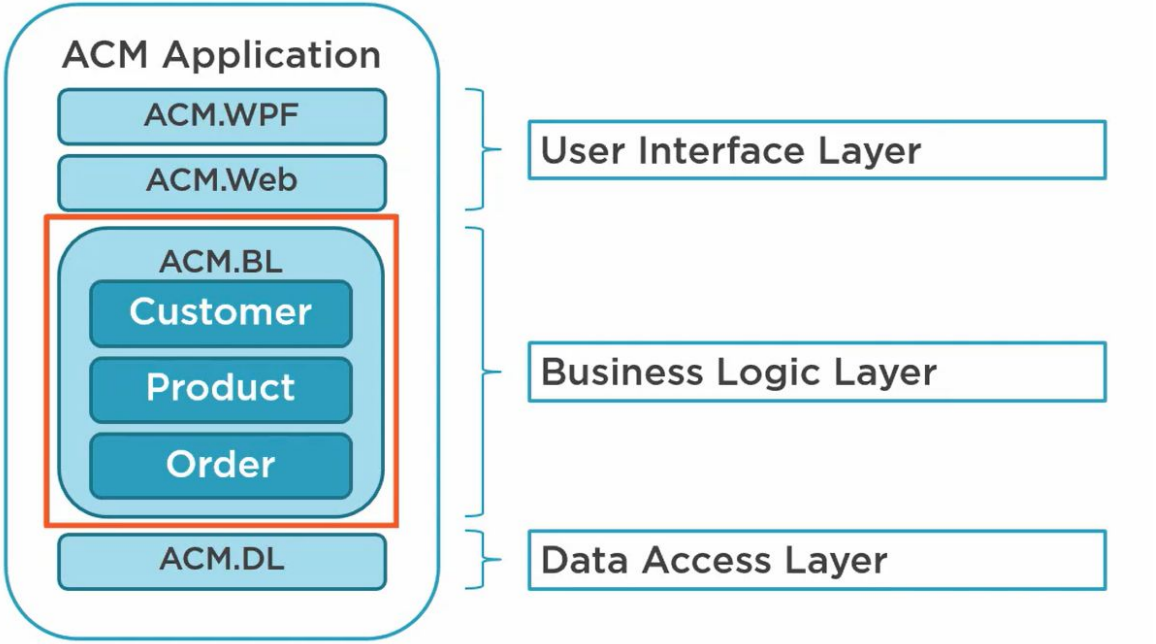
Use in abstract or concrete classes

Optionally overridden by derived class

```
public virtual bool Validate()
{
    ...
}
```


Four Pillars of OOP
Abstraction
Encapsulation
Inheritance
Polymorphism

Componentes Reutilizables



Clases Estáticas

- Clases que no pueden ser instanciadas.
- Se accede a sus miembros usando el nombre de la clase.
- Los miembros de la clase deben ser estáticos.
- Con frecuencia son usadas como un contenedor de métodos útiles.

Creating the method

```
public class StringHandler
{
  public string InsertSpaces(string source)
  { ...
  }
}
```

Using the method

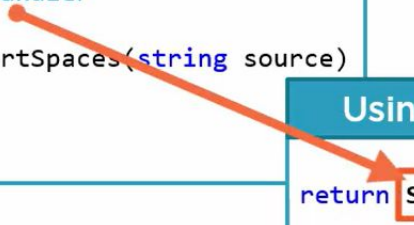
```
var stringHandler = new StringHandler();

return stringHandler.InsertSpaces(productName);
```

Creating the method

```
public static class StringHandler
{
  public static string InsertSpaces(string source)
  { ...
  }
}
```

Using the method

```
return StringHandler.InsertSpaces(productName);
```



Metodos de Extension (Extension Method)

- Agregar métodos a tipos existentes sin modificar el tipo original.
- Agregar métodos a los tipos de .NET.
- Aparecen en Intellisense.
- Solo los métodos estáticos en clases estáticas pueden ser Métodos de Extensión.

Creating the method

```
public static class StringHandler
{
  public static string InsertSpaces(string source)
  { ...
  }
}
```

Using the method

```
return StringHandler.InsertSpaces(productName);
```

Creating the method

```
public static class StringHandler
{
  public static string InsertSpaces(this string source)
  { ...
  }
}
```

Using the method

```
return _productName.InsertSpaces();
```

Interfaces

"In computing, an **interface** is a shared boundary across which two or more separate components of a computer system exchange information.

The exchange can be between software, computer hardware, peripheral devices, humans and combinations of these."

- Wikipedia 1/9/19


User Interface
Web API
Class Interface

Interfaz en C#

```
public interface ILoggable
{
    string Log();
}
```

Resumen POO

- Identificar Clases:
  - Representar las entidades del negocio.
  - Definir Propiedades (datos)
  - Definir métodos (acciones/comportamiento)
- Separar Responsabilidades:
  - Minimizar el acoplamiento
  - Maximizar la cohesión
  - Simplificar el mantenimiento
  - Mejora la testabilidad
- Establecer Relaciones:
  - Definir como los objetos trabajan juntos para llevar a cabo las operaciones de la aplicación
- Aprovechar la reutilización:
  - Involucra extraer lo común
  - Construir clases/componentes reusables
  - Definir interfaces