

1. **Contents**
2. [**The Introspect Framework**](http://ntenhoeve.github.io/#The Introspect Framework)

* [Why the Introspect framework was developed](http://ntenhoeve.github.io/#Why the Introspect framework was developed)
* [Introspect Core Values](http://ntenhoeve.github.io/#Introspect Core Values)
* [Introspect License](http://ntenhoeve.github.io/#Introspect License)
* [Architecture of an introspect application](http://ntenhoeve.github.io/# Architecture of an introspect application)
* [Dependency Injection](http://ntenhoeve.github.io/#Dependency Injection)
* [The Introspect Application](http://ntenhoeve.github.io/#The Introspect Application)
* [Starting a new Introspect Project](http://ntenhoeve.github.io/#Starting a new Introspect Project)

1. [**The Domain Layer**](http://ntenhoeve.github.io/#The Domain Layer)

* [Domain Objects](http://ntenhoeve.github.io/#Domain Objects)
* [Properties](http://ntenhoeve.github.io/#Properties)
* [Action Methods](http://ntenhoeve.github.io/#Action Methods)

1. [**The Service Layer**](http://ntenhoeve.github.io/#The Service Layer)

* [Service Objects](http://ntenhoeve.github.io/#Service Objects)
* [Action Methods](http://ntenhoeve.github.io/#Action Methods)

1. [**The Infrastructure Layer**](http://ntenhoeve.github.io/#The Infrastructure Layer)

* [Infrastructure Objects](http://ntenhoeve.github.io/#Infrastructure Objects)

1. [**The Provider Layer**](http://ntenhoeve.github.io/#The Provider Layer)

* [Provider Objects](http://ntenhoeve.github.io/#Provider Objects)

1. [**The UserInterface Layer**](http://ntenhoeve.github.io/#The UserInterface Layer)

* [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController)

**ActionMethods**

1. [**Object behavior**](http://ntenhoeve.github.io/#Object behavior)

* [Title](http://ntenhoeve.github.io/#Title)
* [Description](http://ntenhoeve.github.io/#Description)
* [Icon](http://ntenhoeve.github.io/#Icon)
* [Visible](http://ntenhoeve.github.io/#Visible)
* [Visible In Form](http://ntenhoeve.github.io/#Visible In Form)
* [Visible In Table](http://ntenhoeve.github.io/#Visible In Table)
* [Enabled](http://ntenhoeve.github.io/#Enabled)
* [Order in Form](http://ntenhoeve.github.io/#Order in Form)
* [Order in Table](http://ntenhoeve.github.io/#Order in Table)
* [Method Execution Mode](http://ntenhoeve.github.io/#Method Execution Mode)
* [Parameter Factory](http://ntenhoeve.github.io/#Parameter Factory)
* [FieldMode](http://ntenhoeve.github.io/#FieldMode)
* [Validation](http://ntenhoeve.github.io/#Validation)
* [MultiLanguage](http://ntenhoeve.github.io/#MultiLanguage)
* [Formatting](http://ntenhoeve.github.io/#Formatting)

1. [**Getting Started**](http://ntenhoeve.github.io/#Getting Started)

* [Integrated Development Environment(IDE)](http://ntenhoeve.github.io/#Integrated Development Environment(IDE))
* [Create your domain object's](http://ntenhoeve.github.io/#Create your domain object's)
* [Creating your Introspect application](http://ntenhoeve.github.io/#Creating your Introspect application)
* [Create a command line application](http://ntenhoeve.github.io/#Create a command line application)
* [Create a desktop application](http://ntenhoeve.github.io/#Create a desktop application)
* [Create a mobile application](http://ntenhoeve.github.io/#Create a mobile application)
* [Create a web application](http://ntenhoeve.github.io/#Create a web application)
* [Create a soap application](http://ntenhoeve.github.io/#Create a soap application)
* [Create a restfull XML application](http://ntenhoeve.github.io/#Create a restfull XML application)
* [Create multiple application types with the same domain model](http://ntenhoeve.github.io/#Create multiple application types with the same domain model)
* [Create a JUnit application](http://ntenhoeve.github.io/#Create a JUnit application)
* [Icons](http://ntenhoeve.github.io/#Icons)
* [Language](http://ntenhoeve.github.io/#Language)
* [Validation](http://ntenhoeve.github.io/#Validation)
* [Authorization](http://ntenhoeve.github.io/#Authorization)
* [Reports](http://ntenhoeve.github.io/#Reports)

1. [**Demo's**](http://ntenhoeve.github.io/#Demo's)
2. [**Downloads**](http://ntenhoeve.github.io/#Downloads)
3. [**Documentation**](http://ntenhoeve.github.io/#Documentation)
4. [**Development**](http://ntenhoeve.github.io/#Development)
5. **The Introspect Framework**

Introspect is a light weight [Java](http://en.wikipedia.org/wiki/Java_(programming_language)) [Application Framework](http://en.wikipedia.org/wiki/Application_framework). It provides a framework for writing business like applications for viewing and editing information in forms and tables (not for graphical application s such as games)  
  
With the introspect framework you only need to create domain classes. Introspect provides you the user interface, without writing any user interface code. This means is that the Introspect Framework is ideal for rapid prototyping or rapid development,or to learn programming Object Orientated Design Or Domain Driven Design (e.g. at schools).  
  
The Introspect framework provides different user interface implementations that you can use for:

* The desktop (based on [Swing](http://en.wikipedia.org/wiki/Swing_(Java)))
* Mobile devices (based on [Android](http://en.wikipedia.org/wiki/Android_(operating_system)))
* The web (based on [Vaadin](http://en.wikipedia.org/wiki/Vaadin))
* [Command line](http://en.wikipedia.org/wiki/Command-line_interface)
* And others

If you want to start coding right now (even if you are a beginner) please go to the [IntrospectGettingStarted](http://ntenhoeve.github.io/#Getting Started) section. If you want to know more about the Introspect Framework keep reading....

1. **Why the Introspect framework was developed**

Almost everyday new libraries, frameworks and tools are being released by the developer community, many of which reinvent the wheel.  
  
This is called the "Yet Another Framework Syndrome" (YAFS), or in more general terms "Not Invented Here" (NIH). While innovation is something we should all welcome, YAFS can lead to confusion and frustration for users because there's a big risk of it introducing more noise than value.  
  
So why did I develop a new framework while there are so many Java application frameworks out there?

1. **Reason 1: Because its fun**

Probably the best reason ever!

1. **Reason 2: Because I wanted to learn**

The journey of developing yet another framework has thought me more than I could have learned implementing an existing framework. Specifically on how other frameworks solved issues that I run into.

1. **Reason 3: Because I wanted it a bit different**

I love the thoughts behind the [Naked Objects Framework](http://nakedobjects.codeplex.com/) (for [.net](http://en.wikipedia.org/wiki/.NET_Framework)) and the [Apache Isis Framework](http://isis.apache.org/) (for [Java](http://en.wikipedia.org/wiki/Java_(programming_language))). But looking at the Apache Isis Framework, there are several things I wanted to do it different in the Introspect framework (good or bad is all open for debate):

* The [domain objects](http://en.wikipedia.org/wiki/Business_logic) and service objects (often repository objects) usually extend a convenience class that contains methods to interact with the Apache Isis framework/ object container. Extending such a class is not mandatory, because you can implement these methods in your objects, but to me this still ignores the principle of "Naked objects" or [POJO’s](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) .
* [Apache Isis](http://isis.apache.org/) uses (depends on) [Maven](http://en.wikipedia.org/wiki/Apache_Maven). Maven has its pros (managing dependencies, and standardizing the build process), but is also famous for its cons (difficult to configure, steep learning curve, etc, etc).
* I personally dislike the way Apache Isis manages the editing of objects. This is very tightly linked to its [persistence framework](http://en.wikipedia.org/wiki/Persistence_framework). I however believe that objects do not necessarily need to be persisted after its been edited. I prefer a different approach: domain objects can be passed to a method as a method parameter. This method parameter can be edited by a user before a method is executed (depending on how the method is annotated). The domain object/ method parameter may then be handled by a [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) like a persistence service (or not at all).

1. **Reason 4: I could not find what I needed.**

I have not found an framework that provides an out of the box User Interface for both desktop, mobile devices, web interface, command line interface, etc. They are probably out there (Naked Objects and Isis coming close) but I haven't found one (or one that I liked).

1. **Introspect Core Values**

Introspect is designed around the following core values:

1. **Based on the Naked Objects Design Pattern**

See the Wiki page on the [Naked Objects Design Pattern](http://en.wikipedia.org/wiki/Naked_objects):

* All [business logic](http://en.wikipedia.org/wiki/Business_logic) should be encapsulated onto the [domain objects](http://en.wikipedia.org/wiki/Business_object_(computer_science)). This principle is not unique to naked objects: it is just a strong commitment to [encapsulation](http://ntenhoeve.github.io/null).
* The [user interface](http://en.wikipedia.org/wiki/User_interface) should be a direct representation of the domain objects, with all user actions consisting, explicitly, of creating or retrieving domain objects and/or invoking [methods](http://en.wikipedia.org/wiki/Method_(computer_science)) on those objects. This principle is also not unique to naked objects: it is just a specific interpretation of an [object-oriented user interface (OOUI)](http://en.wikipedia.org/wiki/Object-oriented_user_interface).
* The user interface should be created 100% automatically from the definition of the domain objects. Introspect uses [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science)) instead of code generation (hence the name of the framework)

1. **Provide a good structure for applications**

* Enforce [separation of concerns](http://en.wikipedia.org/wiki/Separation_of_concerns) (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)).
* Facilitate [domain driven design](http://en.wikipedia.org/wiki/Domain-driven_design).
* The Introspect Framework should not create a [vendor lock-in](http://en.wikipedia.org/wiki/Vendor_lock-in) (not that there is such a thing as an Introspect Framework Vendor, since it is open source). [DomainObject](http://ntenhoeve.github.io/#Domain Objects)'s, [ServiceObject](http://ntenhoeve.github.io/#Service Objects)'s and [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)'s should have no (or almost no) dependencies with the Introspect Framework so that the Introspect Framework could easily (within a few hours) be replaced with another dependency injection framework (such as [Spring](http://en.wikipedia.org/wiki/Spring_Framework), [Guice](http://en.wikipedia.org/wiki/Google_Guice), [Pico container](http://picocontainer.codehaus.org/), etc) or visa versa.
* The introspect framework exists of a few basic interfaces like [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) and [Provider](http://ntenhoeve.github.io/#Provider Objects)s that can have multiple different implementations.

1. **Lightweight**

* The Introspect framework should only be a few kilobytes in size.

1. **Simple to configure**

The Introspect framework has no configuration files (see [“Code or configuration files”](http://ntenhoeve.github.io/null) section in Martin Fowlers article for the arguments why).  
  
There are 3 things that need to be configured:

* Which implementations of the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) and [Provider](http://ntenhoeve.github.io/#Provider Objects)'s are going to be used by the IntrospectApplication.
* Which [ServiceObject](http://ntenhoeve.github.io/#Service Objects) classes and [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) classes are going to be used by the IntrospectApplication.
* The Behavior of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)'s and [ServiceObject](http://ntenhoeve.github.io/#Service Objects) 's (how the domain needs to handled by Introspect framework)

Furthermore the Introspect framework prefers “Configure by Exception”. This means that the Introspect framework uses reasonable defaults wherever possible. These default values can be overridden by the developer.

1. **No dependencies with tools**

* No dependencies with an Integrated Development Environment or build tool.

1. **Introspect License**

Introspect is an [open source](http://en.wikipedia.org/wiki/Open_source) project under the [GNU Lesser General Public License](http://en.wikipedia.org/wiki/GNU_Lesser_General_Public_License). The license can be found [here](http://ntenhoeve.github.io/null).

1. **Architecture of an introspect application**

The Introspect Framework helps you to create a [multi layer architecture](http://en.wikipedia.org/wiki/Multilayered_architecture) for your application. A multi layer architecture has several [design principles](http://martinfowler.com/bliki/LayeringPrinciples.html):

* Separation of concerns (separate responsibilities within the application into different layers).
* Low coupling between layers, high cohesion within them.
* User interface modules should contain no business logic.
* Layers should be testable individual.
* Business logic layers contain no user interface and don't refer to user interface modules.
* No circular references between layers.
* Business layer only uses abstractions of technological services.
* Lower layers should not depend on upper layers.
* Layers should be shy about their internals.
* Layers may share infrastructural aspects (eg security)
* Inbound external interface modules (eg web service handlers) should not contain business logic.

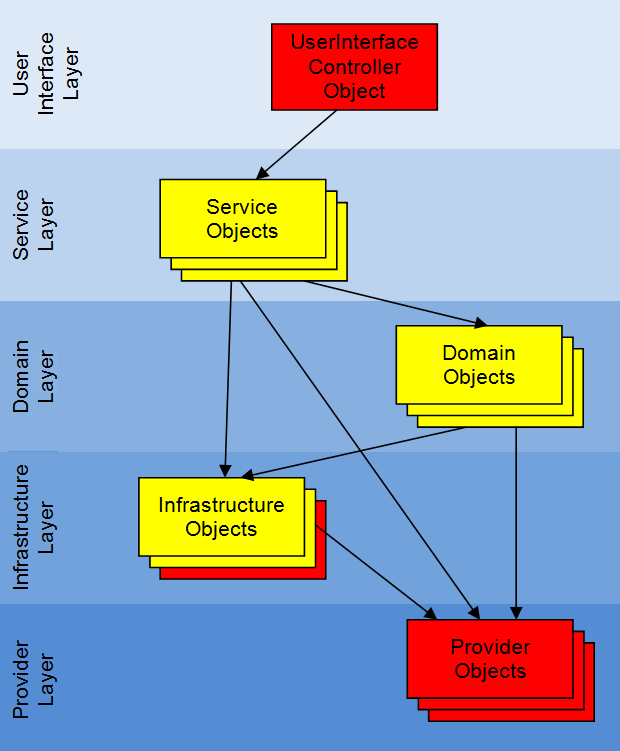
These multi layer architecture design principles try to prevent [spaghetti code](http://en.wikipedia.org/wiki/Spaghetti_code), which is hard to extend, hard to trouble shoot, hard to test and hard to keep [bug](http://en.wikipedia.org/wiki/Software_bug#Etymology) free.

There are many different opinions and definitions on the number of layers, the names of the layers and what each layer should do (see these [examples](https://www.google.nl/search?q=layered+architecture&tbm=isch)). It would be nice if everyone would use the same model and naming. ~~I think that~~ [the definition of layers](http://www.herrodius.com/blog/wp-content/uploads/2010/03/LayeredArchitecture.png) from Eric Evans [Domain Driven Design approach](https://en.wikipedia.org/wiki/Domain-driven_design) is probably the most commonly used. The most important thing is that your multi layered architecture complies with the design principles above.

The [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application) uses the following layer definition, which is pretty close to [Eric Evans Domain Driven Design](https://en.wikipedia.org/wiki/Domain-driven_design) approach.:

* [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer)
* [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) (called application layer by Eric Evans)
* [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer)
* [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer)
* [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer)

Each layer is implemented by an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection)

TODO ADD LINE UIC TO PROVIDERS  
  
  
Red objects are provided by the Introspect framework  
Yellow objects need to be written or included by the developer.

1. **Dependency Injection container**

TODO MOVE DICONTAINER AFTER INTROSPECT APPLICATION AND CREATING A NEW INTROSPECTAPPLICATION

The [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework) is a dependency injection framework and consists of several [dependency injection](http://en.wikipedia.org/wiki/Dependency_injection) containers.

Please read [Martin Fowler](http://en.wikipedia.org/wiki/Martin_Fowler)'s easy to read [article](http://martinfowler.com/articles/injection.html) in which he explains the basics dependency injection.

Each DependencyInjectionContainer is responsible for:

* Creating new instances of types that are registered to a DenpendencyInjectionContainer (via the IntrospectApplication)
* Linking dependencies (references to other objects) to these new instances (using constructor injection)
* Caching these new instances, if we only need one instance (like a singleton)

The [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application) consists of several layers. Each layer has its own [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that is for managing the objects in the same layer:

* [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer): [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) object is managed by a UserInterfaceContainer. The UserInterfaceContainer knows the ServiceContainer
* [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer): [ServiceObject](http://ntenhoeve.github.io/#Service Objects)'s are managed by a ServiceContainer. The ServiceContainer knows the DomainContainer
* [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer): [DomainObject](http://ntenhoeve.github.io/#Domain Objects)'s are managed by a DomainContainer. The DomainContainer knows the InfrastructureContainer
* [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer): [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)'s are managed by a InfrastructureContainer. The DomainContainer knows the ProviderContainer
* [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer): [Provider](http://ntenhoeve.github.io/#Provider Objects) object's are managed by a ProviderContainer.

1. **Constructor Injection**

The [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework) favors constructor injection (see [Martin Fowler](http://en.wikipedia.org/wiki/Martin_Fowler)'s easy to read [article](http://martinfowler.com/articles/injection.html#ConstructorVersusSetterInjection) for the arguments why).

**Injecting dependencies**

All objects within an IntrospectApplication can have references to other objects. These reference objects are injected into an object as constructor parameter during the creation of the object by the DependencyInjectionContainer of that specific layer. This constructor parameter can than be linked to a private final field, so that it can be used throughout the object.

In example:

public class ProductService {

private final ProductRepository productRepository;

public ProductService(ProductRepository productRepository) {

this.productRepository = productRepository;

}

@GenericReturnType(Product.class)

public List<Product> findProduct(ProductSearchCritiria searchCritiria) {

return productRepository.findProduct(searchCritiria);

}

// other action methods...

}

It is good practice to link the constructor parameter (reference object) to a [private](https://en.wikibooks.org/wiki/Java_Programming/Keywords/private) [final](https://en.wikipedia.org/wiki/Final_(Java)#Final_variables) field, so that it is encapsulated and immutable.

If your Object needs a lot of references to other objects (too many constructor parameters), your object has most likely to many responsibilities, which could be solved by splitting up an object.

Note that you can only inject reference object types (use constructor parameters types) that are known to the that specific layer:

* The [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) class can be injected in all objects
* You can only inject objects of the same layer or lower layers (see introspectArchitecture). The introspectframework tries to prevent that objects in the lower layer know (have references to) objects in the higher layers because this is against several design principles of an layered architecture
* You can only inject objects that are registered to the IntrospectApplication by overriding the get...Classes() or get...Class() methods.

1. **The Introspect Application**

The [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) is used as initialization parameter for the Introspect framework.  
An Introspect application has several purposes:

* It provides the name, icon and description of the application (see [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior))
* It provides the location of the distribution package (jar, war)
* It provides the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) type, needed for the application with the [IntrospectApplication.getUserInterfaceControllerClass()](http://ntenhoeve.github.io/#The Introspect Application) method. Each application type (commandline, swing, vaadin) requires different implementations of the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController)
* It provides a list of [ServiceObject](http://ntenhoeve.github.io/#Service Objects) types with the [IntrospectApplication.getServiceClasses()](http://ntenhoeve.github.io/#The Introspect Application) method. [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s basically provide the actionable/menu items
* It provides a list of [DomainObject](http://ntenhoeve.github.io/#Domain Objects) types that needs to be created using dependency injection, with the [IntrospectApplication.getDomainClasses()](http://ntenhoeve.github.io/#The Introspect Application) method.
* It provides a list of [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) types with the [IntrospectApplication.getInfrastructureClasses()](http://ntenhoeve.github.io/#The Introspect Application) method. [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s basically communicate to the outside world (i.e. data base access objects, email clients, soap clients, etc)
* It provides the [Provider](http://ntenhoeve.github.io/#Provider Objects) types with the [IntrospectApplication.get...ProviderClass()](http://ntenhoeve.github.io/#The Introspect Application) methods. [Provider](http://ntenhoeve.github.io/#Provider Objects) Objects help with [cross cutting concerns](https://en.wikipedia.org/wiki/Cross-cutting_concern). Each application type (commandline, swing, vaadin) requires different implementations of the providers

Each application type (command line, Android, Vaadin, etc..) has its own implementation of [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) to help initializing the framework. See the type hierarchy of[IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) to learn which classes can be used and view their java doc to learn how to use them.

If you create a new application you will be extending one of these classes. You will only need to implement some of the methods mentioned above (at least the[IntrospectApplication.getServiceClasses()](http://ntenhoeve.github.io/#The Introspect Application) method).

For examples see the [IntrospectGettingStarted](http://ntenhoeve.github.io/#Getting Started) section.

1. **Starting a new Introspect Project**

How you create a new Introspect application depends on the type of application type that you want to create.

First decide on the type of application you want to create. Then look up the corresponding paragraph in the [IntrospectGettingStarted](http://ntenhoeve.github.io/#Getting Started) section. There you will learn how to create a java project that contains an application class that extends the [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) class. This class will initialize and start the [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework). This class also contains methods that you need to implement to provide the [ServiceObject](http://ntenhoeve.github.io/#Service Objects) classes and [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) classes that are required in the application.

1. **The Domain Layer**

The Domain layer is the hart of any Introspect application. The domain layer represents:

* the concepts of the business
* The business rules
* The state that reflects the business situation   
    
  Because there is many debate on the naming within a layered architecture, the domain layer is sometimes also called:
* [Business](http://en.wikipedia.org/wiki/Business_logic) layer
* [Business logic](http://en.wikipedia.org/wiki/Business_logic) layer
* [Domain model](http://en.wikipedia.org/wiki/Domain_model) layer

The domain layer is basically where all the domain objects are. The DomainContainer is an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that ~~that~~ can be used to create and hold [DomainObject](http://ntenhoeve.github.io/#Domain Objects)'s that need dependency injection.

Note that the [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer) is a middle layer (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)):

* The [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s have NO references to objects in the upper [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer) nor [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer)
* The [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s may have references to the objects in the lower [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) or [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer), but not visa versa!

1. **Domain Objects**

[DomainObject](http://ntenhoeve.github.io/" \l "Domain Objects)s represent entities; the nouns of the domain. If your application domain is a sales application it’s likely that your domain model contains [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s such as: customers, products and orders.

[DomainObject](http://ntenhoeve.github.io/#Domain Objects)s are created by a developer or are reused from an existing application that needs to be re-written. They can be created from scratch or generated from a schema (in example from a [database schema](http://en.wikipedia.org/wiki/Database_schema), [XML schema](http://nl.wikipedia.org/wiki/XML_Schema) or [web service](http://en.wikipedia.org/wiki/Web_Services_Description_Language))

1. **Naming**

[DomainObject](http://ntenhoeve.github.io/#Domain Objects)s names are nouns, such as customer, product and order. They basically describe the things that are important in your application. [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s names need to match the [Ubiquitous Language](http://martinfowler.com/bliki/UbiquitousLanguage.html) (in terms understood by both users and developers).

1. **Construction**

The principle of “naked objects” is that any ['Plain Old Java Object' (POJO)](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) can function as a domain object. In other words: a domain class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

[DomainObject](http://ntenhoeve.github.io/#Domain Objects)s can be created by different objects e.g.:

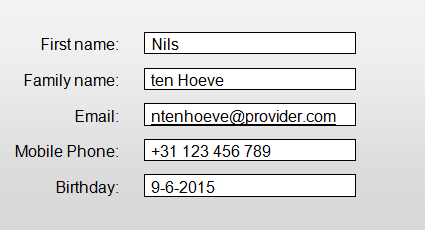
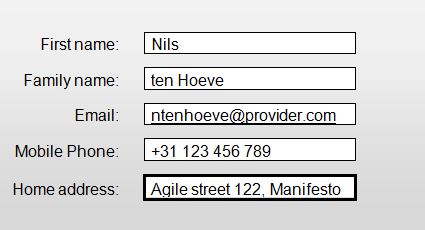
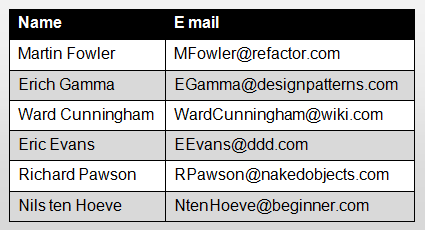
* By a [ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods) such as customerService.createNewCustomer()
* By a [DomainObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)
* By a DomainContainer such as domainContainer.getObject(Customer.class)
* By a [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) method such as orderRepository.allOpenOrders() or shoppingCartFactory.createForCustomer(Customer customer).

There are 2 ways to create new [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s:

* Creating a new [DomainObject](http://ntenhoeve.github.io/#Domain Objects) with the new keyword:  
  In example: Order order=new OrderLine()
* Creating a domain object using Dependency Injection:  
  Sometimes you want a new [DomainObject](http://ntenhoeve.github.io/#Domain Objects) to have references to other objects (being other [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s, [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s or [Provider](http://ntenhoeve.github.io/#Provider Objects)Objects). In example: A Customer object needs a references to a ShoppingCartFactory object. The Customer object can therefore be created by the DomainContainer with Customer customer=(Customer) domainContainer.getObject(Customer.class). The ShopingCart object will automatically be injected as a constructor parameter of the Customer class. In order to create [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s using dependency injection you need to:
  + Add the reference object as a parameter in the constructor and link it to a private field, so that it can be used throughout the class. TODO what happens see above
  + Override the [IntrospectApplication.getDomainClasses()](http://ntenhoeve.github.io/#The Introspect Application) method and return a list of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s that need to be created using Dependency Injection
  + The object that creates the Customer objects needs to have a reference to the DomainContainer. A CustomerService object can get a reference to the DomainContainer when it is created by the ServiceContainer (which is done by the [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework))

1. **Presentation**

An [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) can display domain objects in 3 ways:

* Domain object as form:  
  
* Domain object as a field in a form:  
  
* Domain object as a row in a table:  
  

1. **Domain object members**

Domain objects contain:

* Getter and setter methods (and possible fields): that define [properties](http://en.wikipedia.org/wiki/Property_(programming))
* [DomainObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s: that define user actions
* Methods: that define [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior)
* Annotations: that define [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior)

These members are discussed in more detail in the following paragraphs.

1. **Properties**

DomainObjects have [state](http://en.wikipedia.org/wiki/State_(computer_science)). This means the domainObjects contain information that may change over time. This information is represented by [properties](http://en.wikipedia.org/wiki/Property_(programming)). Here is an example of a domain class customer that has the following properties:

* givenName
* familyName
* fullName
* Todo change to bonusMember

TODO VALIDATE UPPER OR LOWER CASE FOR PROPERTNAMES

public class Customer {

private String givenName;

private String familyName;

private boolean male;

public String getGivenName() {

return givenName;

}

public void setGivenName(String givenName) {

this.givenName = givenName;

}

public String getFamilyName() {

return familyName;

}

public void setFamilyName(String familyName) {

this.familyName = familyName;

}

public String getFullName() {

return new TitleBuilder().append(givenName).append(familyName)

.toString();

}

public boolean isMale() {

return male;

}

public void setMale(boolean male) {

this.male = male;

}

}

Properties are a special type of class members and are an intermediate between [getter and setter methods](http://en.wikipedia.org/wiki/Mutator_method) and a [field](http://en.wikipedia.org/wiki/Field_(computer_science)). The following 3 sections will explain this in more detail.

1. **Getter methods**

* A property always has a getter method, so that other objects can read its value
* The getter method name starts with get, followed by the property name in [CamelCase](http://en.wikipedia.org/wiki/CamelCase) when the property type is NOT a boolean (See the getGivenName() method in the example above)
* The getter method name starts with is, followed by the property name in [CamelCase](http://en.wikipedia.org/wiki/CamelCase) when the property type IS a boolean (See the isMale() method in the example above)
* The getter methods are always public (accessible by other objects)

1. **Setter methods**

* A property might have a setter method, so that other objects can change its value. The property is read-only when it does not have a setter method
* The setter method name starts with set, followed by the property name in [CamelCase](http://en.wikipedia.org/wiki/CamelCase) (See the setFamilyName() method in the example above)
* The setter methods are always public (accessible by other objects)
* The setter methods are commonly placed after the corresponding getter method

1. **Fields**

* Properties can use [fields](http://en.wikipedia.org/wiki/Field_%28computer_science%29) to hold information in the domain object. These fields need to be private (not accessible by other objects) to ensure [encapsulation](http://en.wikipedia.org/wiki/Encapsulation_(computer_science)) (See private fields givenName, familyName and male in the example above)
* A property does not need to have a field. The value can be a computation of fields of other properties (such as the fullName property in the example above), or maybe even a value from an[InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) or a [Provider](http://ntenhoeve.github.io/#Provider Objects).
* Fields are commonly defined at the beginning of the class

1. **Property types**

Property types can be divided into 3 groups:

* Value property: A char, string, boolean, number, date, or other [primitive data type](http://ntenhoeve.github.io/null) s The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display these types as ~~as~~ a [textbox](http://en.wikipedia.org/wiki/Text_box). Assuming that the user is allowed to modify that property, they may enter the value by typing in text, which will be validated and formatted according to the value type. Certainly value types may provide alternative mechanisms for user input, such as a calendar-selector for a date field.
* Enumeration property: The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display these types as as a [combo-box](http://en.wikipedia.org/wiki/Combo_box), containing the different enumeration values as text.
* Reference Property: A 'reference property' or 'entity' is one where the type is another [DomainObject](http://ntenhoeve.github.io/#Domain Objects). Reference properties are thus sometimes referred to as 'associations'. The[UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display a [DomainObject](http://ntenhoeve.github.io/#Domain Objects) property as a [textbox](http://en.wikipedia.org/wiki/Text_box), containing the referenced object (as an icon and title) and a menu button. The menu button will open a [context menu](http://en.wikipedia.org/wiki/Context_menu), that contains ActionMethods to manipulate the property value.
* Collection Property: A 'collection property' is a property that returns an collection of either value properties ([primitive data type](http://ntenhoeve.github.io/null)) or reference properties ([DomainObject](http://ntenhoeve.github.io/#Domain Objects)s). The[UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display these types as a table. If the user clicks on a row a [context menu](http://en.wikipedia.org/wiki/Context_menu) opens, containing ActionMethods to manipulate the property value. Note that you will need to annotate the getter method with a @GenericReturnType annotation in which you need to define the property type. Collections must be initialized (collection properties should never return null). Collection properties should not contain large collections. If you have a big collection associated with a [DomainObject](http://ntenhoeve.github.io/#Domain Objects) it is better to return the collection from anActionMethod in the [DomainObject](http://ntenhoeve.github.io/#Domain Objects), so that the table is displayed on a new tab.

1. **Property behavior**

You can specify certain things about both the behavior and presentation of properties by adding specific attributes or methods. See sections [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior). ~~nilsth~~

1. **Action Methods**

Domain objects can contain action methods (see ActionMethods section)

~~An action is a method in a~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~or~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~that is displayed by the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~as a menu item. An Action method is invoked by the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~when the user clicks on the menu item.~~

TODO MOVE TO MAIN ACTION METHOD SECTION

1. **~~Action Methods for Domain Objects~~**

[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s may have ActionMethods that to do something with or for the given~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~

* ~~These ActionMethods are displayed as menu options in a form tab that represents the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~
* ~~Example: an ShoppingCar object may have an ActionMethod such as checkout().~~

1. **~~Action Methods for Domain Object Properties~~**

~~TODO change naming convention: <actionname><propertyname>~~

[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s may have ActionMethods to do something with the value of a property (e.g. manipulate it).~~

* ~~These ActionMethods are displayed as menu options of a~~[~~DomainObjectProperty~~](http://ntenhoeve.github.io/#Properties)~~in a form tab that represents the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~
* ~~The name of the ActionMethod for a~~[~~DomainObjectProperty~~](http://ntenhoeve.github.io/#Properties)~~must begin with the action name, followed by its property name~~

~~Examples:~~

* ~~A ShoppingCar object may have ActionMethod such as addLineItems(LineItem lineItem) or removeLineItem(LineItem lineItem) or removeAllLineItems().~~
* ~~A Customer object may have ActionMethod such as moveToNewAddress(Address new Address).~~

1. **~~Action Methods Convention~~**

~~Any method in a~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~or~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~can be an Action method, provided that it complies with the following convention:~~

* ~~The method has no parameter or a single~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~parameter~~
* ~~Its return type (if any) are types recognized by the Introspect framework (see below)~~
* ~~The method is not a getter method or a setter method (see~~[~~DomainObjectProperty~~](http://ntenhoeve.github.io/#Properties)~~)~~
* ~~The method is not a BehavioralMethod~~
* ~~The method is public (not private)~~
* ~~The method is NOT static~~

1. **~~Action method names~~**

~~The name of an ActionMethod should describe the action and match the~~[~~Ubiquitous Language~~](http://martinfowler.com/bliki/UbiquitousLanguage.html)~~(in terms understood by both users and developers). Keep in mind that the goal of a user is almost never to create, update or delete objects. Method names like: createPerson, updatePerson and removePerson should therefore be avoided where possible. Method names like registerNewBirth, registerMarriage, registerPersonDeceased would be better names.~~

1. **~~Action method parameter~~**

~~An action method either has no parameter or a single~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~as a parameter. If not, the Introspect Framework will not recognize a method as an ActionMethod.~~

1. **~~Action method return value~~**

~~The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~renders the output of a method, depending on the type of the action method return value:~~

* ~~No return value (void method): The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will display a short message when the method has been executed~~
* [~~A primitive data type~~](http://ntenhoeve.github.io/null)~~: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will display a message dialog that displays the return value after the method is been executed~~
* ~~A~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~in a form on a new tab.~~
* ~~A~~[~~collection~~](http://en.wikipedia.org/wiki/Java_collections_framework)~~of~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s in table on a new tab. Note that you will need to annotate the Action method with a @GenericReturnType annotation in which you need to define the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~type.~~
* ~~A URI: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the contents of the URI on a new tab.~~
* ~~A DownloadStream: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will open a "Save as" dialog" so that the file can be down loaded.~~

1. **~~Action execution modes~~**

~~ActionMethods can be annotated, so that the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~knows how the ActionMethod needs to be invoked after the user has clicked on the corresponding menu item.~~

* ~~@ExecutionMode(ExecutionModeType.EXECUTE\_METHOD\_DIRECTLY ): executes the method directly without user intervention~~
* ~~@ExecutionMode( ExecutionModeType.EXECUTE\_METHOD\_AFTER\_CONFORMATION): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a confirmation dialog. The method is executed after the user activates the confirmation button. The method is NOT executed when the user cancels the confirmation dialog.~~
* ~~@ExecutionMode( ExecutionModeType.EDIT\_PARAMETER\_THAN\_EXECUTE\_METHOD\_OR\_CANCEL): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a form on a new tab, so that the user can modify (edit) the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~. The method is executed with the edited~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~as the method parameter, when the user clicks the confirmation button on the button bar. The method is NOT executed when the user clicks on cancel in the bottom bar.~~

1. **~~Action behavior~~**

~~You can specify certain things about both the behavior and presentation of actions methods, such as their display name, icon, visibility, enabled state, etc with help of annotations or behavior methods. See sections on~~[~~ObjectBehavior~~](http://ntenhoeve.github.io/#Object behavior)~~. DomainLayer IntrospectArchitecture~~

1. **The Service Layer**

The [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) (sometimes also called [application layer](http://ntenhoeve.github.io/application%20layer)) gives the user access to the [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s so that the user can work on them.

The ServiceContainer is an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that represents the [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) and holds and manages [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s.

Note that the [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) is a middle layer (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)):

* The [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s have NO references to objects in the [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer)
* The [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s may have references to the objects in the lower [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer), [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) and [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer), but not visa versa!

1. **Service Objects**

The word 'service' implies:

* There is a client that needs to be served (for the IntrospectFramework this is the user, trough the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController))
* There are activities\ operations (for the IntrospectFramework this is defined as ActionMethod's)
* An activity\ operation is done with other things ( for the IntrospectFramework this are ActionMethod parameters and return values should be [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s)

Eric Evans explains in his book [Domain Driven Design](https://en.wikipedia.org/wiki/Domain-driven_design), that a good [ServiceObject](http://ntenhoeve.github.io/#Service Objects) has three characteristics:

* The operation relates to a domain concept that is not a natural part of a [DomainObject](http://ntenhoeve.github.io/#Domain Objects) (Entities or Value Objects).
* The interface is defined in terms of other elements of the domain model.
* The operation is state-less

These characteristics are discussed in more detail in the following paragraphs.

1. **Service objects operations relates to a domain concept that is not a natural part of a DomainObject**

[ServiceObject](http://ntenhoeve.github.io/#Service Objects)s contain [ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s. The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) displays these methods as menu items and invokes these methods once a user clicks on the menu item.[ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s create or retrieve [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s where the user does not have an existing [DomainObject](http://ntenhoeve.github.io/#Domain Objects) to navigate from.

1. **Service objects define an interface in terms of the domain model**

[ServiceObject](http://ntenhoeve.github.io/#Service Objects)s basically provides the user (via the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController)) access to [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s, so that the user can work on them (via the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController)).

1. **Service objects are state-less**

Quoiting Eric Evans: “ [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s should be state-less. State-less here means that any client can use any instance of the [ServiceObject](http://ntenhoeve.github.io/#Service Objects)s without regard to the instance’s individual history. The execution of the service will use information that is accessible globally, and may even change that global information (have side-effects). But it does not hold state of its own that affects its behavior, as most domain objects do.”

[ServiceObject](http://ntenhoeve.github.io/#Service Objects)s do not have state and therefore should not have properties (no getter and setter methods). My personal opinion is that a [ServiceObject](http://ntenhoeve.github.io/#Service Objects) with state is a [code-smell](http://en.wikipedia.org/wiki/Code_smell), which you can solve by moving the [ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s that share state (fields) to ~~an~~ new or existing [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s.

1. **Service Objects should be flat**

[Object Orientated Programming](https://en.wikipedia.org/?title=Object-oriented_programming) favors to put business logic and the validation logic into the [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s (and sometimes [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s) as much as possible.[ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s should therefore not contain business logic or validation logic, but delegate the work to collaborations of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s and [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) s ,in order to prevent the [Anemic Domain Model](http://martinfowler.com/bliki/AnemicDomainModel.html) - [anti-pattern](http://en.wikipedia.org/wiki/Anti-pattern).

1. **A web shop example**

* The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) class calls the [ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods) findProduct(searchCriteria) method on [ServiceObject](http://ntenhoeve.github.io/#Service Objects): ProductService
* This method will call the findProduct on the [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects): ProductRepository
* This method will return a list of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s that meet the search criteria
* The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) displays the found list of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s as a table in a new tab.

1. **Naming**

[ServiceObject](http://ntenhoeve.github.io/#Service Objects)s are normally named after the [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s that they service and have the suffix 'Service' (e.g. CustomerService, OrderService, etc).

1. **Construction**

The principle of “naked objects” is that any ['Plain Old Java Object' (POJO)](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) can function as a [ServiceObject](http://ntenhoeve.github.io/#Service Objects). In other words: a service class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

[ServiceObject](http://ntenhoeve.github.io/#Service Objects)s are instiantated by the [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework), and therefore need to be registered to the [IntrospectApplication.getServiceClasses()](http://ntenhoeve.github.io/#The Introspect Application) method.

In example:

public class WebShop extends IntrospectApplicationFor... {

@Override

public List<Class<?>> getServiceClasses() {

List<Class<?>> serviceClasses = new ArrayList<>();

serviceClasses.add(ProductService.class);

serviceClasses.add(ShoppingCartService.class);

serviceClasses.add(OrderService.class);

return serviceClasses;

}

// etc...

}

ServiceObjects can have references to other objects. These objects are injected into the ServiceObject (see the DependencyInjectorContainer section.

[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~s can have references to other objects. These reference objects are injected into the~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~as constructor parameter during the creation of the~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~by the~~[~~IntrospectFramework~~](http://ntenhoeve.github.io/#The Introspect Framework)~~(by the ServiceContainer). This constructor can than be linked to a private final field, so that it can be used throughout the~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~.~~

~~In example:~~

~~public class ProductService {~~

~~private final ProductRepository productRepository;~~

~~public ProductService(ProductRepository productRepository) {~~

~~this.productRepository = productRepository;~~

~~}~~

~~@GenericReturnType(Product.class)~~

~~public List<Product> findProduct(ProductSearchCritiria searchCritiria) {~~

~~return productRepository.findProduct(searchCritiria);~~

~~}~~

~~// other action methods...~~

~~}~~

~~Note that you can only inject reference object types (use constructor parameters types) that are known to the~~[~~IntrospectFramework~~](http://ntenhoeve.github.io/#The Introspect Framework)~~.~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~s can be injected with the following types (see also~~[~~IntrospectArchitecture~~](http://ntenhoeve.github.io/# Architecture of an introspect application)~~):~~

* ~~The~~[~~IntrospectApplication~~](http://ntenhoeve.github.io/#The Introspect Application)~~class~~
* ~~Types registered to the~~[~~IntrospectApplication.getServiceClasses()~~](http://ntenhoeve.github.io/#The Introspect Application)~~method~~
* ~~Types registered to the~~[~~IntrospectApplication.getDomainClasses()~~](http://ntenhoeve.github.io/#The Introspect Application)~~method~~
* ~~Types registered to the~~[~~IntrospectApplication.getInfrastructureClasses()~~](http://ntenhoeve.github.io/#The Introspect Application)~~method~~
* ~~The~~[~~Provider~~](http://ntenhoeve.github.io/#Provider Objects)~~classes~~

~~Note that it is good practice to link the constructor parameter (reference object) to a~~[~~private~~](https://en.wikibooks.org/wiki/Java_Programming/Keywords/private)[~~final~~](https://en.wikipedia.org/wiki/Final_(Java)#Final_variables)~~field, so that it is encapsulated and immutable.~~

~~Note that that if your~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~needs a lot of references to other objects (too many constructor parameters), your~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~has most likely to many responsibilities, which could be solved by splitting up a~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~.~~

1. **~~Presentation~~**

~~TODO main menu's, object menu's, property menu's~~

~~The MainMenu will display all actionMethods of all registered ServiceObjects that can directly be executed (without the need of an opened DomainObject). This means that only ActionMethods that either have no method parameter or have an parameterFactory are displayed as menu items in the main menu.~~

1. **Service object members**

Service objects contain:

* [ServiceObjectActionMethod](http://ntenhoeve.github.io/#Action Methods)s: that define user actions
* Methods: that define [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior)
* Annotations: that define [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior)

These members are discussed in more detail in the following paragraphs.

1. **Action Methods**

ServiceObjects are all about ActionMethods that represent the main menu items (see section ActionMethods)

~~An action is a method in a~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~or~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~that is displayed by the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~as a menu item. An ActionMethod is invoked by the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~when the user clicks on the menu item.~~

1. **~~Action Methods for Service Objects~~**

[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~s always have one ore more ActionMethods.~~

* ~~These ActionMethods are displayed by the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~as menu options in the main menu or in a form tab that represents the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~
* ~~Examples: an CustomerService object may have an ActionMethod such as findCustomer(CustomerSearchArgument searchArgument).~~

[~~Object Orientated Programming~~](https://en.wikipedia.org/?title=Object-oriented_programming)~~favors to put business logic and the validation logic into the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s (and sometimes~~[~~InfrastructureObject~~](http://ntenhoeve.github.io/#Infrastructure Objects)~~s) as much as possible.~~[~~ServiceObjectActionMethod~~](http://ntenhoeve.github.io/#Action Methods)~~s should therefore not contain business logic or validation logic, but delegate the work to collaborations of~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s and~~[~~InfrastructureObject~~](http://ntenhoeve.github.io/#Infrastructure Objects)~~s, in order to prevent the~~[~~Anemic Domain Model~~](http://martinfowler.com/bliki/AnemicDomainModel.html)~~-~~[~~anti-pattern~~](http://en.wikipedia.org/wiki/Anti-pattern)~~.~~

1. **~~Action Methods Convention~~**

~~Any method in a~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~or~~[~~ServiceObject~~](http://ntenhoeve.github.io/#Service Objects)~~can be an Action method, provided that it complies with the following convention:~~

* ~~The method has no parameter or a single~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~parameter~~
* ~~Its return type (if any) are types recognized by the Introspect framework (see below)~~
* ~~The method is not a getter method or a setter method (see~~[~~DomainObjectProperty~~](http://ntenhoeve.github.io/#Properties)~~)~~
* ~~The method is public (not private)~~
* ~~The method is NOT static~~

1. **~~Action method names~~**

~~The name of an ActionMethod should describe the action and match the~~[~~Ubiquitous Language~~](http://martinfowler.com/bliki/UbiquitousLanguage.html)~~(in terms understood by both users and developers). Keep in mind that the goal of a user is almost never to create, update or delete objects. Method names like: createPerson, updatePerson and removePerson should therefore be avoided where possible. Method names like registerNewBirth, registerMarriage, registerPersonDeceased would be better names.~~

1. **~~Action method parameter~~**

~~An action method either has no parameter or a single~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~as a parameter. If not, the Introspect Framework will not recognize a method as an ActionMethod.~~

1. **~~Action method return value~~**

~~The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~renders the output of a method, depending on the type of the action method return value:~~

* ~~No return value (void method): The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will display a short message when the method has been executed~~
* [~~A primitive data type~~](http://ntenhoeve.github.io/null)~~: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will display a message dialog that displays the return value after the method is been executed~~
* ~~A~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~in a form on a new tab.~~
* ~~A~~[~~collection~~](http://en.wikipedia.org/wiki/Java_collections_framework)~~of~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~s in table on a new tab. Note that you will need to annotate the Action method with a @GenericReturnType annotation in which you need to define the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~type.~~
* ~~A URI: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~displays the contents of the URI on a new tab.~~
* ~~A DownloadStream: The~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~will open a "Save as" dialog" so that the file can be down loaded.~~

1. **~~Action execution modes~~**

~~ActionMethods can be annotated, so that the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~knows how the ActionMethod needs to be invoked after the user has clicked on the corresponding menu item.~~

* ~~@ExecutionMode(ExecutionModeType.EXECUTE\_METHOD\_DIRECTLY ): executes the method directly without user intervention~~
* ~~@ExecutionMode( ExecutionModeType.EXECUTE\_METHOD\_AFTER\_CONFORMATION): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a confirmation dialog. The method is executed after the user activates the confirmation button. The method is NOT executed when the user cancels the confirmation dialog.~~
* ~~@ExecutionMode( ExecutionModeType.EDIT\_PARAMETER\_THAN\_EXECUTE\_METHOD\_OR\_CANCEL): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a form on a new tab, so that the user can modify (edit) the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~. The method is executed with the edited~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~as the method parameter, when the user clicks the confirmation button on the button bar. The method is NOT executed when the user clicks on cancel in the bottom bar.~~

1. **~~Action behavior~~**

~~You can specify certain things about both the behavior and presentation of actions methods, such as their display name, icon, visibility, enabled state, etc with help of annotations or behavior methods. See section on~~[~~ObjectBehavior~~](http://ntenhoeve.github.io/#Object behavior)~~.~~

1. **The Infrastructure Layer**

The [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) contains [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)'s that provide generic technical capabilities to support the higher layers.

The [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) is also know as:

* [Data access layer](http://en.wikipedia.org/wiki/Data_access_layer)
* [Persistence layer](http://en.wikipedia.org/wiki/Persistence_layer)
* Logging Layer
* Networking Layer
* And other services which are required to support the [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) or [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer)

The InfrastructureContainer is an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that represents the [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) and holds and manages [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s.

Note that the [InfrastructureLayer](http://ntenhoeve.github.io/#The Infrastructure Layer) is a middle layer (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)):

* The [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s have NO references to objects in the upper [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer), [ServiceLayer](http://ntenhoeve.github.io/#The Service Layer) nor [DomainLayer](http://ntenhoeve.github.io/#The Domain Layer)
* The [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s may have references to the objects in the lower [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer), but not visa versa!

1. **Infrastructure Objects**

[InfrastructureObject](http://ntenhoeve.github.io/" \l "Infrastructure Objects)s provide generic technical capabilities to support the higher layers. [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s communicate with other systems such as external hardware (like disks), databases, web-services, etc...

1. **Function and naming of InfrastructureObjects**

The name of an [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s depends on what it does. It is common practice to use the following naming:

* Communicating with a [database](https://en.wikipedia.org/wiki/Database):  
  class names end with 'Repository', e.g.: CustomerRepository, OrderRepository, etc...
* Communicating with a [web-service](http://ntenhoeve.github.io/null):  
  class names end with 'Client', e.g.: EmailClient, GoogleMapsClient, SOAPClient, etc..
* Creating objects with a [factory](https://en.wikipedia.org/wiki/Factory_(object-oriented_programming)) class:  
  class names end with 'Factory', e.g.: ReportFactory, etc..
* [Logging](https://en.wikipedia.org/wiki/Logfile):  
  class names end with 'Logger'

1. **Construction**

The principle of “naked objects” is that any ['Plain Old Java Object' (POJO)](http://en.wikipedia.org/wiki/Plain_Old_Java_Object) can function as a [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects). In other words: a [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects) class does not have to inherit from any special class, nor implement any particular interface, nor have any specific attributes.

You can:

* write new [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s.
* reuse [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s from existing projects
* use or extend one of the [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s of one of the IntrospectInfrastructure projects. Open the type hierarchy of the [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects), to see all these convenience classes
* or use or extend [InfrastructureObject](http://ntenhoeve.github.io/#Infrastructure Objects)s from other (open source) projects

InfrastructureObjects can have references to other objects. These objects are injected into the InfrastructureObjects (see the DependencyInjectorContainer section.

1. **Infrastructure Object Presentation**

The methods of infrastructure object are unknown to the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) and are not displayed on the [User Interface](https://en.wikipedia.org/wiki/User_interface).

1. **The Provider Layer**

The [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer) contains [Provider](http://ntenhoeve.github.io/#Provider Objects) objects that provide generic [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework) capabilities ([cross cutting concerns](http://ntenhoeve.github.io/cross%20cutting%20concerns)) to support the higher layers (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application))

The ProviderContainer is an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that represents the [ProviderLayer](http://ntenhoeve.github.io/#The Provider Layer) and holds and manages [Provider](http://ntenhoeve.github.io/#Provider Objects)Objects.

Note that this layer is the bottom layer (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application) ), which means that objects in the upper layers may haved references to [Provider](http://ntenhoeve.github.io/#Provider Objects)Objects, but not visa versa!

1. **Provider Objects**

[Provider](http://ntenhoeve.github.io/" \l "Provider Objects)s are responsible for different [cross cutting concerns](http://ntenhoeve.github.io/cross%20cutting%20concerns) within a IntrospectApplication such as

* Authorization (see AuthorizationProvider)
* Validation (see ValidationProvider)
* Multi language (see LanguageProvider)
* Notifications (see NotificationProvider)
* File path information (see PathProvider)
* Reflection information (see ReflectionProvider)
* Version information (see AboutProvider)

[Provider](http://ntenhoeve.github.io/#Provider Objects)s may be used by any class within an application.

Providers are interfaces and can have multiple implementations (depending on what type of application you are using/writing). Which implementation of each provider needs to be used within an application is defined in the [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) class. You are free to implement your own [Provider](http://ntenhoeve.github.io/#Provider Objects) implementation and register it by overwriting one of the[IntrospectApplication.get...ProviderClass()](http://ntenhoeve.github.io/#The Introspect Application) methods

1. **Provider Construction**

[Provider](http://ntenhoeve.github.io/#Provider Objects)'s are instantiated by the ProviderContainer (see DependencyInjectionCobtainer) ProviderObjects can have references to other ProciderOobjects. These objects are injected into the ProviderObjects (see the DependencyInjectorContainer section.

**Provider Presentation**

The methods of [Provider](http://ntenhoeve.github.io/#Provider Objects) object are not displayed by the UserInterfaceController.

Todo remove nils ten hoeve

1. **Authorization Provider**

TODO PROVIDERS NEED TO BE H2 CHAPTERS SO THAT THEY WILL SHOW UP IN THE INDEX

[Authorization](http://ntenhoeve.github.io/null)

, means the ability to control what an individual user can see and do within an application, based upon their identity and the role(s) assigned to them.

The AuthorizationProvider facilitates this with methods:

* AuthorizationProvider.getCurrentUserName()
* AuthorizationProvider.userInRole(String userRole)

1. **Types of AuthorizationProvider**

There can be different types of AuthorizationProvider implementations, depending on the authorization mechanism you prefer. In example: you could write an implementation that uses:

* ~~the DefaultAuthorizationProvider (always returns true on the .userInRole(String) method)~~
* hard coded authorization
* file based authorization
* database authorization
* web container authorization (e.g. [Apache Tomcat Realm](https://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html))
* [LDAP](https://nl.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol)
* [Active directory](https://nl.wikipedia.org/wiki/Active_Directory)
* or other

TODO add example of a hard coded AuthorizationProvider

1. **Registering an AuthorizationProvider**

The AuthorizationProvider is registered to the framework with the [IntrospectApplication.getAuthorizationProviderClass()](http://ntenhoeve.github.io/#The Introspect Application) method. By default it returns the DefaultAuthorizationProvider, which always returns true on the .userInRole(String) method. You can register another AuthorizationProvider implementation by overriding the [IntrospectApplication.getAuthorizationProviderClass()](http://ntenhoeve.github.io/#The Introspect Application)method.

1. **Using an AuthorizationProvider**

The AuthorizationProvider is used by the @Hidden and @Disabled annotations (see ObjectBehaviour). If you want to use the AuthorizationProvider in your code you need to inject it into your object (see DependencyInjectionContainer)

1. **Validation Provider**

The UserinterfaceController sometimes let’s the user edit an DomainObject before it is passed as an ActionMethod parameter (depending on how the ActionMethod is annotated, see ExecutionMode ).

The UserinterfaceController then validates the edited DomainObject using the ValidationProvider before the ActionMethod is called.The ValidationProvider will use validationAnnotations and ValidationMethods that are located in the DomainObjects them selves to validate the domain objects.

**ValidationProvider implementation.**

There are many validation libraries and frameworks available. The IntrospectFramework complies with the JSR??? by using the ???? library, combined with its own behaviouralMethods.

You are free to implement or extend your own implementation and register it to your IntrospectApplication by overriding the IntrospectApplication.getValidationProviderClass () method

1. **Language Provider**

The LanguageProvider makes multilanguage support possible. The framework supports the English language by default because these texts are embedded in the code as part of the ambiguous language (it does not make sense to mix an other language into your code).

**Language property files**

Texts for other languages are stored in property files. The name of the need to have the following file name:

TODO VERIFY FORMAT

<application configuration folder>/Resources\_<Language\_code>.propertiesproperties

* <application configuration folder> These files need to be located at the configuration folder of the application. See PathProvider for more information
* <Language\_code> two letter language code (see java Locale ....)

The language property files contain key value pairs.

The key is an reference to part of the code

The value is the translated text

Keys:

Keys are made up by a FullName followed by .displayname or.description

TODO INSERT fullName

Examples of key value lines in property files:

com.acme.order.ShoppingCart.displayname=Winkel wagen

com.acme.product.ProductService.displayname=Producten

Todo: how to use special characters

**Getting the translated text**

The UserInterfaceController will call the Language provider to try and get the appropriate text from the language property files (depending on the selected language in the graphical userinterface). If it can’t find this value it will display the default values (in English)

TODO CODE EXAMPLE OF VALIDATION TEXTS OR EXCEPTION TEXTS

TODO EXPLAIN HOW THE LANGUAGE IS SELECTED

TODO HOW TO SET DEFAULT TEXT OTHER THAN DEFAULT

TODO EXPLAIN HOW TO GENERATE OR UPDATE LANGUAGE PROPERTY FILES WITH A SPECIAL INTROSPECT APPLICATION

1. **Notification Provider**

TODO What it does

TODO How you can use it (or explain who uses it)

TODO Code example

TODO How to register a custom ... provider

1. **Path Provider**

The PathProvider provides the locations of files that are needed within an IntrospectApplication.

The actual file location depends on the type of introspectApplication used.

TODO explain interface

TODO explain the different file locations for swing, android, vaadin

The PathProvider is used by the user interfaceController and can als be used in other objects:

TODO Code example file path to database in infrastructureobject

You can provide your own implementation of the pathProvider.

You can register your customizer  PathProvider implementation by overriding the [IntrospectApplication.getPathProviderClass()](http://ntenhoeve.github.io/#The Introspect Application) method.

**Reflection Provider**

The ReflectionProvider provides information on objects (ClassInfo), properties (PropertyInfo) of actionMethod (ActionMethodInfo) using reflection. The UserinterfaceController uses this information to know how the user interface should look like and behave.

You can use the ReflectionProvider in your code when you need meta information.

TODO Code example

1. **About Provider**

The AboutProvider provides the following information on all objects used in an introspect application:

* The class name
* The package name (jar,war,etc...)
* The version of the jar file (from the meta info file)
* The authors of the jar file (from the meta info file)

This about information can be displayed on a graphical user interface.

1. **The UserInterface Layer**

The [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer) is also know as presentation layer (although I think that 'presentation layer' is a poor name, because it is responsible for so much more). The [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer) contains the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController), which is responsible for showing information to the user and processing the information from the user using the objects in the lower layers (see[IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)).

The UserInterfaceContainer is an [IntrospectContainer](http://ntenhoeve.github.io/#Dependency Injection) that represents the [UserInterfaceLayer](http://ntenhoeve.github.io/#The UserInterface Layer) and holds and manages the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController).

Note that this layer is the top layer, which means it may know all the objects in the lower layers but not visa versa! See [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)

1. **UserInterfaceController**

The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) is responsible for showing information to the user and processing the information from the user using the objects in the lower layers (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)).

An user interface could be any type of user interface, in example:

1. **Command line interface**

* [Command line interface](http://en.wikipedia.org/wiki/Command-line_interface) is a text-based user interface where the user (or client) issues commands in the form of successive lines of text (command lines).
* The user can be a person or another computer application (using a shell).
* See IntrospectApplicationForCommandLine.

1. **Desktop interface**

* A desktop interface is a [graphical user interface](http://en.wikipedia.org/wiki/Graphical_user_interface) for a computer with an [desktop environment](https://en.wikipedia.org/wiki/Desktop_environment)
* The user is likely to be a person
* Introspect has an desktop implementation based on ~~Swing for a computer using~~ [Swing](http://ntenhoeve.github.io/). See IntrospectApplicationForSwing

1. **Mobile interface**

* A mobile interface is a [graphical user interface](http://en.wikipedia.org/wiki/Graphical_user_interface) for mobile devices such as smart phones and tablets.
* The user is likely to be a person
* Introspect will have a mobile implementation using Android. See IntrospectApplicationForAndroid

1. **Web interface**

* A [web application](https://en.wikipedia.org/wiki/Web_application) is a [graphical user interface](http://en.wikipedia.org/wiki/Graphical_user_interface) for [web browsers](https://en.wikipedia.org/wiki/Web_browser)
* The user is likely to be a person
* Introspect will have a mobile implementation using Vaadin. See IntrospectApplicationForVaadin

1. **SOAP interface**

* The SOAP interface is an web service that allows other computer applications to interact via the [Simple Object Access Protocol (SOAP)](http://en.wikipedia.org/wiki/SOAP)
* The user is likely to be another computer application
* Introspect might have a SOAP implementation in the future

1. **RESTfull XML interface**

* The RESTfull XML interface is an web service that allows other computer applications to interact using [Representational State Transfer (RESTfull)](http://en.wikipedia.org/wiki/Representational_state_transfer) with [XML](https://en.wikipedia.org/wiki/XML)
* The user is likely to be another computer application
* Introspect might have a RESTfull XML implementation in the future

1. **RESTfull JSON interface**

* The RESTfull SJON interface is an web service that allows other computer applications to interact using [Representational State Transfer (RESTfull)](http://en.wikipedia.org/wiki/Representational_state_transfer) with [JSON](https://en.wikipedia.org/wiki/JSON)
* The user is likely to be another computer application
* Introspect might have a RESTfull XML implementation in the future

1. **JUnit interface**

* The JUnit interface is an interface to test objects in te lower layers (see [IntrospectArchitecture](http://ntenhoeve.github.io/# Architecture of an introspect application)), using the [JUnit test framework](https://en.wikipedia.org/wiki/JUnit)
* The user is the JUnit test framework
* See IntrospectApplicationForJUnit.

Please see the class hierarchy of the [IntrospectApplication](http://ntenhoeve.github.io/#The Introspect Application) class to find all the different user interface implementations. See GettingStarted section with specific instructions for each type of introspect application

1. **Action Methods**

Todo action method as main section

An action is a method in a [DomainObject](http://ntenhoeve.github.io/#Domain Objects) or [ServiceObject](http://ntenhoeve.github.io/#Service Objects) that is displayed by the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) as a menu item. An Action method is invoked by the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) when the user clicks on the menu item.

1. **Action Methods for Domain Objects**

[DomainObject](http://ntenhoeve.github.io/#Domain Objects)s may have ActionMethods that to do something with or for the given [DomainObject](http://ntenhoeve.github.io/#Domain Objects).

* ~~These ActionMethods are displayed as menu options in a form tab that represents the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~
* Example: an ShoppingCar object may have an ActionMethod such as checkout().

1. **Presentation**

ActionMethods of DomainObjects are displayed as menu items in the DomainObjectMenu.

**DomainObject menu**

TODO PICTURE Domain object MENU and table row context menu

1. The **DomainObjectMenu is displayed on the toolbar of a formview (that displays a DomainObject) or as a context menu when the user clicks on a row of a table view. The DomainObjectMenu allows a user to preform an action on or with a DomainObject. An DomainObjectMenu contrails all the actionMethods of the DomainObject and all actionmethod s of ServiceObjects that take the DomainObject as a parameter. Each ServiceObject is displayed as a sumenu**
2. **Action Methods for Domain Object Properties**

TODO change naming convention: <actionname><propertyname>

[DomainObject](http://ntenhoeve.github.io/#Domain Objects)s may have ActionMethods that do something with the value of a property (e.g. manipulate it).

* ~~These ActionMethods are displayed as menu options of a~~[~~DomainObjectProperty~~](http://ntenhoeve.github.io/#Properties)~~in a form tab that represents the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~.~~
* The name of the ActionMethod for a [DomainObjectProperty](http://ntenhoeve.github.io/#Properties) must begin with the action name, followed by its property name

Examples:

* A ShoppingCar object may have an property lineItems thar has a propertyActionmethods such as addLineItems(LineItem lineItem) or removeLineItem(LineItem lineItem) or removeAllLineItems().
* A Customer object may have an property adress that has a PropertyActionMethod such as moveToNewAddress(Address new Addres)

1. **Presentation**

ActionMethods of DomainObjects are displayed as menu items in the DomainObjectMenu.

**Property menu**

TODO PICTURE property context MENU and field table row context menu

1. The **PropertyMenu is displayed as a context menu in a formview when the menu button of a referenceProperty or a row of a collectionproperty is clicked. The propertyMenu allows a user to manipulate a DomainObjectProperty. It contains all propertyactionmethods of and all actionmethod s of ServiceObjects that take the DomainObject as a parameter. Each ServiceObject is displayed as a sumenu**

**The Action Methods for Service Objects**

ServiceObjects always have one ore more ActionMethods.

• These ActionMethods are displayed by the UserInterfaceController as menu options in the main menu or in a form tab that represents the DomainObject.

1. • Examples: an CustomerService object may have an ActionMethod such as findCustomer(CustomerSearchArgument searchArgument).
2. **Presentation**

ActionMethods of serviceObjects can appear as menu items in the mainMenu, DomainObjectMenu, or PropertyMenu

**Main menu**

TODO PICTURE MAIN MENU

The main menu is displayed on a panel on the left hand side of the user interface (sometimes this panel is hidden bur it can always be made visible again using the main toolbar. The main menu provides the user access to the DomainObject s so that the user can work on them. The main menu contains all ActionMethods of ServiceObject’s

that either have no method parameter or have an parameterFactory.

ActionMethods of serviceObjects that take a DomainObject as a parameter and have no parameterFactory are displayed in DomainObjectMenu s and propertyMenus.

Each servicObject is always displayed as a sub menu

1. **Action Methods Convention**

Any method in a [DomainObject](http://ntenhoeve.github.io/#Domain Objects) or [ServiceObject](http://ntenhoeve.github.io/#Service Objects) can be an Action method, provided that it complies with the following convention:

* The method has no parameter or a single [DomainObject](http://ntenhoeve.github.io/#Domain Objects) parameter
* Its return type (if any) are types recognized by the Introspect framework (see below)
* The method is not a getter method or a setter method (see [DomainObjectProperty](http://ntenhoeve.github.io/#Properties))
* The method is not a BehavioralMethod
* The method is public (not private)
* The method is NOT static

1. **Action method names**

The name of an ActionMethod should describe the action and match the [Ubiquitous Language](http://martinfowler.com/bliki/UbiquitousLanguage.html) (in terms understood by both users and developers). Keep in mind that the goal of a user is almost never to create, update or delete objects. Method names like: createPerson, updatePerson and removePerson should therefore be avoided where possible. Method names like registerNewBirth, registerMarriage, registerPersonDeceased would be better names.

1. **Action method parameter**

An action method either has no parameter or a single [DomainObject](http://ntenhoeve.github.io/#Domain Objects) as a parameter. If not, the Introspect Framework will not recognize a method as an ActionMethod.

1. **Action method return value**

The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) renders the output of a method, depending on the type of the action method return value:

* No return value (void method): The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display a short message when the method has been executed
* [A primitive data type](http://ntenhoeve.github.io/null): The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will display a message dialog that displays the return value after the method is been executed
* A [DomainObject](http://ntenhoeve.github.io/#Domain Objects): The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) displays the [DomainObject](http://ntenhoeve.github.io/#Domain Objects) in a form on a new tab.
* A [collection](http://en.wikipedia.org/wiki/Java_collections_framework) of [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s: The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) displays the [DomainObject](http://ntenhoeve.github.io/#Domain Objects)s in table on a new tab. Note that you will need to annotate the Action method with a @GenericReturnType annotation in which you need to define the [DomainObject](http://ntenhoeve.github.io/#Domain Objects) type.
* A URI: The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) displays the contents of the URI on a new tab.
* A DownloadStream: The [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) will open a "Save as" dialog" so that the file can be down loaded.

1. **Action execution modes**

ActionMethods can be annotated, so that the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) knows how the ActionMethod needs to be invoked after the user has clicked on the corresponding menu item. For more information see the ExecutionMode

* ~~@ExecutionMode(ExecutionModeType.EXECUTE\_METHOD\_DIRECTLY ): executes the method directly without user intervention~~
* ~~@ExecutionMode( ExecutionModeType.EXECUTE\_METHOD\_AFTER\_CONFORMATION): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a confirmation dialog. The method is executed after the user activates the confirmation button. The method is NOT executed when the user cancels the confirmation dialog.~~
* ~~@ExecutionMode( ExecutionModeType.EDIT\_PARAMETER\_THAN\_EXECUTE\_METHOD\_OR\_CANCEL): the~~[~~UserInterfaceController~~](http://ntenhoeve.github.io/#UserInterfaceController)~~opens a form on a new tab, so that the user can modify (edit) the~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~. The method is executed with the edited~~[~~DomainObject~~](http://ntenhoeve.github.io/#Domain Objects)~~as the method parameter, when the user clicks the confirmation button on the button bar. The method is NOT executed when the user click~~s on cancel in the bottom bar.

1. **Action behavior**

You can specify certain things about both the behavior and presentation of actions methods, such as their display name, icon, visibility, enabled state, etc with help of annotations or behavior methods. See sections on [ObjectBehavior](http://ntenhoeve.github.io/#Object behavior). DomainLayer IntrospectArchitecture

1. **Object behavior**

The IntrospectApplication, ServiceObjects and DomainObjects can have behavior that defines how the objects act or how they are displayed. Behavior can be defined with:

* methods that are recognized by the [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework)
* annotations that are recognized by the [IntrospectFramework](http://ntenhoeve.github.io/#The Introspect Framework)

TO-DO RECOGNIZED METHODS (see naked objects doc) TO-DO RECOGNIZED ANNOTATIONS (TODO verify if we missed a sections by inspecting ClassInfo and MethodInfo and PropertyInfo)

**Display Name**

The class names, domainObjectproperty names and actionmethod names are part of the ambiqiuos language (in terms both understand by users and developers) and should therefore match both the code and user interface.

**DisplayName Default**

Class names, DomainObjectProperty names and ActionMethod names in the code Base use names such as OrderService, orderLines, addOrderLine (using no spaces, camel case and no special characters)

If the user is a an human, more user friendly names are needed such as “Orders” “Order lines” and “Add order line”.

The IntrospectFramework will therefore convert these names to a human readable format when needed.

**DisplayName for ServiceObjects**

It is recommended that the DisplayName of ServiceObject ‘s is the plural form of the DomainObject that the ServiceObject represents. In example: The displayName of CustomerService is “Customers”.

TODO IMPLEMENT:

The IntrospectFramework creates the plural DisplayName for serviceObjects by removing the “Service” suffix and append an “s” at the end (or replace the last “y” with “ies”). Use the DisplayName annotation if the default plural form is incorrect

**DisplayName Annotation**

In some cases the default DisplayName does not suffices, in example when:

* A different use of capital case is needed
* Special characters are needed that can not be used in the code
* The plural form of the default displayName of a ServiceObject is incorect

In these cases you can use the @DisplayName annotation before the class keyword, before the getter method of a property or before the ActionMethod.

TODO EXAMPLE ACMEWebShop

Note that the @DisplayAnnotation is intended for the English language only. Use the DisplayName for multilanguage for other languages

**DisplayName for multilanguage**

The framework supports DisplayNames for multiple languages. The UserInterfaceController calls the LanguageProvider to get the displayNames from a language specific property file. For more information see LanguageProvider.

1. **Title**

DomainObjects that have identity (entities) need to have a dynamic title that help users to distinguish objects of the same type (e.g. Type customer versus “John Doe”). This title should exist of all the properties that identify the object. The title is therefore dynamic: it changes when the value of these properties change.

In example: The title of a customer could be a customer number, followed by the given name, followed by the family name. If the customer changes it’s name, than so does the title (but not its identity)

TODO is there a title annotation if so document it and inherit doc

**Default title**

The introspectframework provides a default title based on the properties that are normally displayed in tables. This is a best guess. It is therefore recommended to always implement the toString method

**Title (toString) method**

Define the title by overriding the the toString() method of your DomainObject

(TODO example with Customer toString and TitleBuilder)

**TitleBuilder**

Often an DomainObject can be identified by a single property. The toString() method can than simply return this property value to return the title.

TODO IMPLEMENT:

TODO INSERT DOCUMENTATION FROM TITLEBUILDER

In other cases a DomainObject is identified by multiple property values. In this case it is recommended to use the StringBuilder (see the example above). The TitleBuilder is an extension of the StringBuilder, but has some additional functionality such as

* The “append” methods will append a separator and a given value. You can use the default separator (a space) or use an “append” method where you give the separator as a first parameter followed by the value.
* The “contact” methods will append a given value without a separator
* Both “append” and “contact” methods will ignore null values or reference objects that do not have a toString implementation
* Both “append” and “contact” methods have methods where you can specify format values (e.g. date, time and numbers)

(TODO titlebuilder + example)

**Description**

Each DomainObjectProperty or ActionMethod can have a text to explain the class member more detail. This description is often displayed in a graphical user interface when the user hovers over the property or action menu.

**Description Default**

By default the description is the same as the DisplayName, but than it does not provide any additional information.

**Description Annotation**

TODO INSERT From ANNOTATION JAVADOC

The description can be defined with an annotation:

Syntax: @Description (string description)

TODO EXAMPLE

**Description Method**

Or the description can be defined with an method recognized by the ReflectFramework (allows dynamic behaviour)

Syntax: description <class name, propertName, or actionMethodName>()

TODO EXAMPLE

**Description Multilanguage**

The framework supports descriptions in multiple languages. The UserInterfaceController calls the LanguageProvider to get the descriptions from a language specific property file. For more information see LanguageProvider.

1. **Icon**

IntrospectApplication objects, ServiceObjects and DomainObjects can have icons that are displayed in graphical user interfaces. These icons help the user to quickly identify what they are looking at.

**Icon Default**

The IntrospectFramework will try to find an image with the same class name or actionmethod name but with the “.png” extension in the same package.

* Filename signature for object icon: <package>/<className>.png (e.g. c:/MyProject/src/customer/service/CustomerService.png)
* TODO VERIFY:Filename signature for ActionMethod icon: <package>/<className>\_<actionMethodName>.png (e.g. c:/MyProject/src/customer/service/CustomerService\_findCustomer.png)

Note: no icon will be displayed when the “<className>.png” file can not be found.

Note that currently the PNG image format is the only format that is supported. Adding more image types will slow down lookup performance.

**Icon annotation**

TODO INSERT From ANNOTATION JAVADOC

If you do not want to use the default icon, you can define an alternative icon by placing an Icon annotation before the “class” key word or before the ActionMethod.

Syntax: @Icon (iconName)

Parameter iconname: name of the alternative PNG file that represents the icon. This can be one of the PNG images of the Introspect Library or a PNG file you have put in the applications image folder (see PathProvider )

TODO EXAMPLE of class icon (User png from introspect lib) and actionmethod icon

(EditCustomer.png from image folder)

**Icon method**

Instead of the icon annotation you can also define the icon with a method recognized by the ReflectFramework. This allows you to change the icon dynamically during runtime, based on state (e.g. when the DomainObject Person is a male or female).

Syntax: public string icon<className or actionMethodName>()

TODO EXAMPLE

**Hidden**

TODO Insert documentation from file

TODO IMPLEMENT

DomainObjectProperties or ActionMethods visible on the userinterface by default,but in some cases you may want to hide them, e.g.:

* Because not all information needs to be displayed in the userinterface (e.g. a database id key or version number for optimistic locking)
* Because you want to limit the number of visible properties in a table (e.g. to limit the number of columns in a table to prevent the table to become cluttered with to much information)
* Depending on the state of a domain object you do not want to display inrelevant information
* Depending on the state of a domain object you do not want a user to activate an action method (e.g. hide an actionmethod “submit” once the domainObject already is submitted)
* Because a user is not authorised to see a property value
* Because a user is not authorised to activate an actionmethod

**Hidden annotation**

TODO INSERT From ANNOTATION JAVADOC

You can hide a DomainObjectProperty by putting the @Hidden annotation before the getter method or you can hide an ActionMethod by putting the @Hidden annotation before the ActionMethod.

Syntax: @Hidden (for, exceptForUsers, exceptForUserRoles)

Parameters:

* For: optional and for properties only: to indicate if the property should be hidden in FormViews, TableViews or both.
* ExceptForUsers: optional comma separated string of users names that are allowed to see the DomainObjectProperty or ActionMethod
* exceptForUserGroupers: optional comma separated string of user roles that are allowed to see the DomainObjectProperty or ActionMethod

**Hidden method**

TODO IMPLEMENT

You can also hide a DomainObjectProperty or DomainObjectActionMethod depending on the DomainObject state (the value of its properties).

The @Hidden annotation and the hidden method have a logical AND function. Both need to be true in order to hide the domainObjectProperty or ActionMethod.

Syntax: public boolean hidden<property name or actionMethodName>()

ReturnValue: a boolean that is true if the domainObjectProperty or ActionMethod needs to be hidden

1. **Disabled**

TODO Insert documentation from file

TODO IMPLEMENT

DomainObjectProperties or ActionMethods are enabled on the userinterface by default, but in some cases you may want to disable them. If something is disabled they are grayed out on the user interface. Disabled properties can not be edited (readonly) and disabled ActionMethod s can not be invoked.

You may want to disable items when:

* Not all properties need to be editable on the userinterface but the code might need a setter method to set the propery value when the DomainObject is created from the database.
* Depending on the state of a DomainObject you do not want the user to edit certain properties
* Depending on the state of a DomainObject you do not want a user to invoke an ActionMethod (e.g. prevent invoking actionmethod “submit” once the domainObject already is submitted)
* Because a user is not authorised to edit a property value
* Because a user is not authorised to invoke an actionmethod

Note that if an user is not authorized to change a [DomainObjectProperty](http://ntenhoeve.github.io/#Properties) or call an ActionMethod it is best to hide the method or property instead of disabling it. In general you do not want to confuse users (clutter the userinterface) with options that they are not allowed to use anyway.

**Disabled annotation**

TODO INSERT From ANNOTATION JAVADOC

You can disable a DomainObjectProperty by putting the @disabled annotation before the getter method or you can disable an ActionMethod by putting the @Disabled annotation before the ActionMethod.

Syntax: @Disabled ( exceptForUsers, exceptForUserRoles)

Parameters:

* ExceptForUsers: optional comma separated string of users names that are allowed to see the DomainObjectProperty or ActionMethod
* exceptForUserGroupers: optional comma separated string of user roles that are allowed to see the DomainObjectProperty or ActionMethod

**Disable method**

TODO IMPLEMENT

You can also disable a DomainObjectProperty or DomainObjectActionMethod depending on the DomainObject state (the value of its properties).

The @Disabled annotation and the disabled method have a logical AND function. Both need to be true in order to disable the domainObjectProperty or ActionMethod.

Syntax: public boolean disabled<property name or actionMethodName>()

ReturnValue: a boolean that is true if the domainObjectProperty or ActionMethod needs to be disabled

1. **Order**

TODO IMPLEMENT

TODO INSERT From ANNOTATION JAVADOC

The order in which the class members ( the properties and ActionMethods) are displayed can be defined with the @Order annotation. Note that the order of class members can not be changed during runtime.

The @Order annotation can be added before the getter method of a property or before an ActionMethod. Both Properties and ActionMethod s are ordered separately but use the same annotation.

Syntax: @Order (double sequenceNumber)

The sequenceNumber of the @Order annotation determines the position of the class member. The classmember with the lowest sequenceNumber will be shown first, a higher sequenceNumber later. Class members that are not annotated will be shown last.

It is recommended to use an interval (of let's say 10) between the sequenceNumbers so that you do not have to renumber all the existing @Order annotations every time you add a new class member in between. Otherwise you can always fall back on using decimals (e.g. 1.5 or 10.25).

TODO EXAMPLE

**Format**

TODO INSERT From ANNOTATION JAVADOC

Some property types such as date, time, number can be formatted with help of the Format annotation. Note that the format can not be changed during runtime.

The @Format annotation can be added before the getter method of a property.

Syntax: @Format (string pattern)

Please see the JavaDoc of the ... formatters to learn more about the patterns that can be used.

TODO find out which formatters

TODO example

1. **FieldMode**

TODO INSERT From ANNOTATION JAVADOC

Properties are rendered by the Userinterface layer as fields when the user edits a DomainObject in a form view. With the FieldMode you can select some alternative options for these fields. Note that the format can not be changed during runtime.

Syntax: @FieldMode (FieldModeType fieldModeType)

* FieldModeType.Multiline: renders a text field to edit a string with multiple lines
* FieldModeType.Password: renders a field to edit a password string (obscuring the input so that other viewers can not read the secret password)
* FieldModeType.richTextField: renders a field to edit a string, with formatting options such as bold, underline, italic,etc...)

TODO Verify all FieldModeType s and add missing ones

TODO EXAMPLE

1. **ExecutionMode**

TODO INSERT From ANNOTATION JAVADOC

ActionMethods can be annotated, so that the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) knows how the ActionMethod needs to be invoked after the user has clicked on the corresponding menu item. Note that the ExecutionMode can not be changed during runtime.

Syntax: @ExecutionMode(ExecutionModeType executionModeType )

* @~~ExecutionMode~~(ExecutionModeType.EXECUTE\_METHOD\_DIRECTLY ): executes the method directly without user intervention
* @~~ExecutionMode~~( ExecutionModeType.EXECUTE\_METHOD\_AFTER\_CONFORMATION): the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) opens a confirmation dialog. The method is executed after the user activates the confirmation button. The method is NOT executed when the user cancels the confirmation dialog.
* @~~ExecutionMode~~( ExecutionModeType.EDIT\_PARAMETER\_THAN\_EXECUTE\_METHOD\_OR\_CANCEL): the [UserInterfaceController](http://ntenhoeve.github.io/#UserInterfaceController) opens a form on a new tab, so that the user can modify (edit) the [DomainObject](http://ntenhoeve.github.io/#Domain Objects). The method is executed with the edited [DomainObject](http://ntenhoeve.github.io/#Domain Objects) as the method parameter, when the user clicks the confirmation button on the button bar. The method is NOT executed when the user clicks on cancel in the bottom bar.

1. **Parameter Factory**

The parameter factory allows you to create an object for an ActionMethod. This object can then be edited by the user (depending how the actionmethod is annotated, see ExecutionMode) after which it is passed as method parameter when the actionMethod is invoked.

The MainMenu will display all actionMethods of all registered ServiceObjects that can directly be executed (without the need of an opened DomainObject). This means that only ActionMethods that either have no method parameter or have an parameterFactory are displayed as menu items in the main menu.

**ParameterFactory annotation**

When adding the @ParameterFactory Annotation before an actionMethod, the UserIntterfaceController will first try to create a new DomainObject. This object can then be edited by the user (depending how the ActionMethod is annotated, see ExecutionMode) after which it is passed as method parameter when the actionMethod is invoked.

Syntax: @ParameterFactory

TODO EXAMPLE find customer with paramfactory annotation

**ParameterFactory method**

When adding the @ParameterFactory method (normally located after an actionMethod), the UserIntterfaceController will first get a new DomainObject from the ParameterFactoryMethod. This object can then be edited by the user (depending how the ActionMethod is annotated, see ExecutionMode) after which it is passed as method parameter when the actionMethod is invoked.

Syntax: public<domainObject type> prameterFactory<actionMethodName>()

TODO EXAMPLE OF ordersWithinPeriod METHOD

1. **Validation**

The UserinterfaceController sometimes let’s the user edit an DomainObject before it is passed as an ActionMethod parameter (depending on how the ActionMethod is annotated, see ExecutionMode ).

The UserinterfaceController then validates the edited DomainObject using the ValidationProvider before the ActionMethod is called.The ValidationProvider will use validationAnnotations and ValidationMethods that are located in the DomainObjects them selves to validate the domain objects.

**Validation annotations**

Properties can be validated by putting validation annotations before the getter method of a property. Validation annotations are often used for the more basic validations. The following sections will explain the specific validation annotations that can be used.

**Validation methods**

Properties and DomainObjects can also be validated with validation methods located in the domain class so that you can do more complicated validation using code.

ValidationMethods conventions:

* Syntax: validate<propertyname or businessrulename). E.g. for property: validateStartDate() or for business rule: validateCustommerAlreadyExists()
* Method may not have a method parameter
* Method must return a ValidationResult
* Method must be public
* Method may not be static
* May not change the state of the DomainObject

The following sections will explain the specific ValidationMethods that can be used.

**TYPES OF VALIDATIONS**

There are different types of validation

* Is the property mandatory?
* Has the property the correct format ?
* Is the property within the correct range?
* Does the DomainObject adhire to the defined business rules?

**Is the property mandatory?**

If a property is mandatory (must have a value) you can add one of the following annotations:

@NotNull

@NotEmpty

TODO EXAMPLE

**Has the property the correct format?**

Some properties require a certain format (such as property types: date, time and currency). This format can be defined by adding the following annotations:

@Regexp

@Mask

TODO EXAMPLE

**Is the property within the correct range?**

Add one of the following annotations if a property needs to be within a predefined range:

@GreatherThan

TODO SEE JSR??? TO FIND ALL ANNOTATIONS

TODO EXAMPLE

**Does the DomainObject adhire to the defined business rules?**

Business rules are needed when:

* Validation is more complex and can not be handled with ValidationAnnotations, but needs to be done with code.
* Validation requires multiple property values (e.g. start date must be before end date)
* Validation requires a different source in order to validate (e.g. check a repository if a name if unique, or ask the LanguageProvider for the localised format)

BusinessRules are defined by ValidationMethods.

TODO EXAMPLE

TODO SEE JSR AND SEE IF WE ARE MISSING ANNOTATIONS

TODO EXPLAIN VALIDATION RESULT

1. **TODO REMOVE FORMATTING SECTION**

TODO REMOVE LANGUAGE SECTION

1. **Getting Started**

This section is a how-to guide and explains all you need to know to create and deploy your own applications.

1. **Integrated Development Environment(IDE)**

Get your favourite IDE installed on your computer. If you are a newby in developing applications, you could try and install Eclipse. It is a great free IDE that you can download from the internet, plus Eclipse is used in all instruction videos, so if you do not have a preference for an IDE, I recommend starting with Eclipse. There are also other free IDE’s available that will do just fine. Next step is to download the Introspect Libraries from GIT into your IDE (TODO youtube video)

1. **Create your domain object's**

TODO

1. **Creating your Introspect application**

With introspect you can create different types of applications right out of the box: • Create a command line application • Create a desktop application • TODO Create a mobile application • TODO Create a web application • TODO Create a soap application • TODO Create a restfull XML application TODO Each application type has it’s own “getting started” video.

1. **Create a command line application**

(TODO youtube video)

1. **Create a desktop application**

(TODO youtube video)

1. **Create a mobile application**

(TODO youtube video)

1. **Create a web application**

(TODO youtube video)

1. **Create a soap application**

(TODO youtube video)

1. **Create a restfull XML application**

(TODO youtube video)

1. **Create multiple application types with the same domain model**

(TODO youtube video)

1. **Create a JUnit application**

See IntrospectApplicationForAndroid (TODO youtube video)

1. **Icons**

The Introspect user interface supports icons (see section on icons) or see the following tutorial video (TODO youtube video)

1. **Language**

The Introspect user interface supports multi languages (see section on LanguageProvider) or see the following tutorial video (TODO youtube video)

1. **Validation**

The Introspect user interface supports multi languages (see section on LanguageProvider) or see the following tutorial video (TODO youtube video)

1. **Authorization**

(TODO youtube video)

1. **Reports**

(TODO youtube video)

1. **Demo's**

TODO

1. **Downloads**

Source Code:

* [Git Hub Web Pages](https://github.com/ntenhoeve/Introspect-Framework)
* [Download Introspect projects as zip](https://github.com/ntenhoeve/Introspect-Framework/archive/master.zip)

1. **Documentation**

The documentation of the IntrospectFramework is generated from its JavaDoc (starting with the ReflectDocumentation file) and released in different formats and fora with help of the SoftwareDocumentationGenerator:

* [Documentation as Git Hub Wiki](https://github.com/ntenhoeve/Introspect-Framework/wiki)
* [Documentation as Web Page](http://ntenhoeve.github.io/)

**Development**

* [Issue Tracker](https://github.com/ntenhoeve/Introspect-Framework/issues)
* [Roadmap](https://github.com/ntenhoeve/Introspect-Framework/milestones)

TODO PUT ON GITHUB ROADMAP

Reflect 1.0

* Change title default behaviour (service object to plural – service + ‘s or y to ies)
* Improve title builder zie naked objects of isis
* Change visible to hide and
  + for FORMVIEW, TABLEVIEW, FORM\_AND\_TABLE\_VIEW
  + ExceptForUsers (comma separated String)
  + ExceptForUserRoles (comma separated String)
  + and remove visible in for and visible in table annotations and methods
* Reflect methods: <reflectname><action method or propertyname>
* Merge order in table and orderinform into order
* Complete doc
* Rename Introspect to Reflect

Eclipse order members tool?

ReflectForCommandLine

ReflectForJunit

ReflectForSwing

ReflectForAndroid

ReflectForLanguageFiles

ReflectForSoap

ReflectForRestfullXML

ReflectForRestfullJSON

Logging all user actions support

Autocompletion for fields

Options for fields

Custom fields

Custom views

Different datastore support

Different authorization support

* [About the developer](https://nl.linkedin.com/in/nilstenhoeve)