# A Brief Introduction to Protégé and Reasoners

A question you rightfully may be pondering is: Why translate object oriented classes into OWL? The answer is that it can help you to find logical inconsistencies in your class designs. In this post I will introduce the tools that will eventually enable you to find logical inconsistencies in your class designs.

The tool we will use is called Protégé. Download and installations instructions for Protégé can be found at <https://protegewiki.stanford.edu/wiki/Install_Protege5>.

In this post I will provide two screencasts:

1. In the first screencast I will show you how to enter the OWL representation of the <code>Person</code> class introduced in the previous post.
2. In the second screencast I will show you how to run a reasoner and how an inconsistency can arise.

On to the first screencast:

1. Create a <code>Person</code> class.
2. Create the data properties.
   1. <code>name</code>
   2. <code>surname</code>
   3. <code>age</code>
3. Through sub-classing state that the <code>Person</code> class necessarily have a
   1. <code>name</code>,
   2. <code>surname</code> and
   3. <code>age</code>.
4. If we run the reasoner on this ontology, no inconsistencies will be found.

In the second screencast I show how an inconsistency can arise. The steps are as follows:

1. Create an individual called <code>sarah</code> of type <code>Person</code>.
2. Run the reasoner. You will see the reasoner give no errors. This may come as a surprise to you since we have not set the <code>name</code>, <code>surname</code> or <code>age</code> data properties for the individual called <code>sarah</code>. In OWL this behaviour is expected due to what is called the **open world assumption**. OWL makes no assumption with regards to knowledge that is not stated explicitly. Since we did not state that the <code>sarah</code> individual does not have, for example, a <code>name</code>, the reasoner found no error in our ontology. This is different from typical database behaviour where absence of information is often assumed to indicate that the information does not exist, which is referred to as the closed world assumption.
3. Now let us change our <code>sarah</code> individual to state that it does not have a <code>name</code>. This is achieved by stating that the <code>sarah</code> individual is of type <code>name max 0 xsd:string</code>. This states that the <code>sarah</code> individual can have a maximum of 0 <code>name</code> data properties of type <code>xsd:string</code>.
4. If we run the reasoner now it shows that we have an inconsistency. We can ask Protégé to explain the inconsistency.
5. The explanation states that <code>sarah</code> is of type <code>Person</code> and of type <code>name max 0 xsd:string</code>. But <code>Person</code> is a subclass of <code>name some xsd:string</code>. This states that individuals of type <code>Person</code> must have at least 1 <code>name</code> property of type <code>xsd:string</code>. Hence, the reason for the inconsistency.

Admittedly this example is contrived: there is not much sense in creating a <code>Person</code> class which we state must have a <code>name</code> and then create an individual of type <code>Person</code> which we then state does not have a <code>name</code>. But this was done here to show you how to use a reasoner to find inconsistencies in your ontology and to show you what information you can expect when your ontology is inconsistent.