Customizing Classes

A class is an aggregation of properties and operations that describes a complex data type from which objects can be created. You can configure classes as described on Changing the Attributes of Data Model Elements.

Attribute	Description
General	Specify a general class. Generalizations define that one class inherits properties and operations from another class. A generalization is the relationship from the child element (the more specific element, a subclass) to the parent element (the more general element, a super class). The child class is fully consistent with the parent class, and provides additional information.
Interfaces	Specify an interface to derive operations from. In contrast to a class, an interface has no properties nor implementations. Interfaces are used to define common operations of multiple classes, and then derive from that interface. Operations of interfaces do not have an implementation but only define the signature (parameters and types).
Stereotypes	You can apply stereotypes to a class to be able to access more configuration options, e. g. to control XML serialization.

Instances of a class can be created using the create statement. Classes are not only identified by their name but also by their path in the data model. This said, **Implementation.Order.Customer** is a different class than **Implementation.Quote.Customer**.

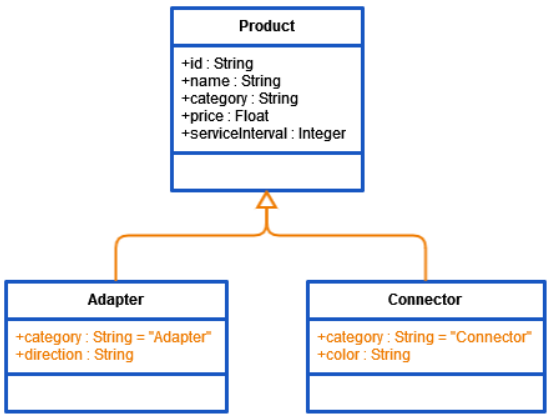
On this Page:

Related Pages:

- Customizing Properties
- Changing the Attributes of Data Model Elements
- XML Serialization
- Stereotypes

Generalization Example

Our **ACME** example company sells **adapters** and **connectors**. Both are product types of the company that share some properties but do also have dedicated properties that apply to the distinct type:



Adapters have a direction and connectors have a color (both marked in red in the example class diagram above). **The category of adapters is "Adapter", the category of connectors is "Connector". Apart from that they share the common properties of a product like id, name, category, price and serviceInterval.**

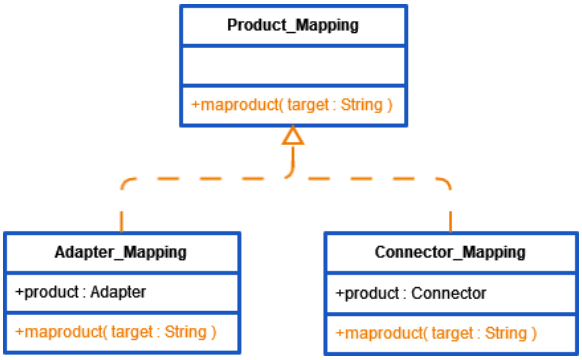
Creating an instance of **Adapter** would result in the following object:

Property	Value
id	NULL
name	NULL
category	Adapter
price	NULL
serviceInterval	NULL

direction	NULL

Interface Example

The **ACME** example company has a library that uses the following **Product_Mapping** interface.



The interface operation **mapProduct(target : String)** is implemented by two classes: **Adapter_Mapping** and **Connector_Mapping**. Depending on which of the two types is provided to the library, the library uses a different implementation of **mapProduct**.

Troubleshooting

As mentioned before, classes are not only identified by their name but also by their path in the data model. In some cases this does not apply, e.g. when objects are serialized to the persisted event queue. When this queue is parsed, only the name of the class is taken into account. This can lead to compiler errors like the following:

```
class "urn:<path to the class>.aClass" does not support operation
"anOperation$$1912144410"
```

This error occurs if the class **aClass** has been defined twice in your model, having the same name and namespace but with different paths in the data model. The compiler cannot distinct both classes in this case and randomly pics one of them to look for the related operation.

To avoid this you can

1. Rename one of the classes.
2. Apply a divergent namespace to one of the classes using the stereotype **XML** (refer to Controlling the XML Serialization With Stereotypes for more information on this stereotype).