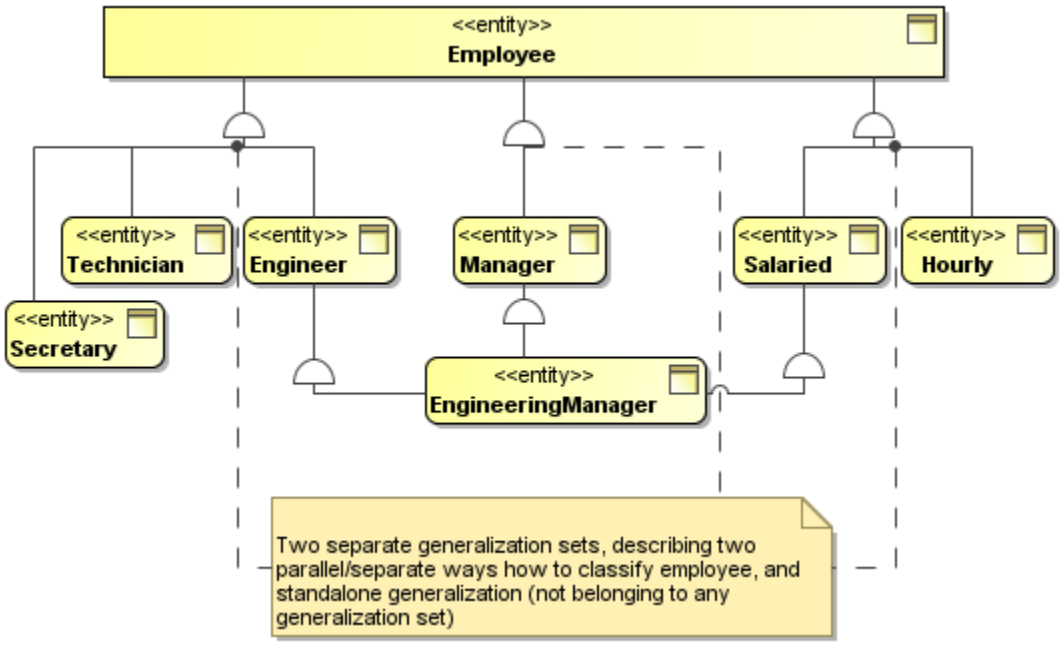
# Generalization and specialization

An ER diagram has a support for generalization / specialization modeling. Generalization and Specialization are really the same relationship, with a different direction of classification. Generalization is bottom-up, while specialization is top-down. Thus, they use the same model element.

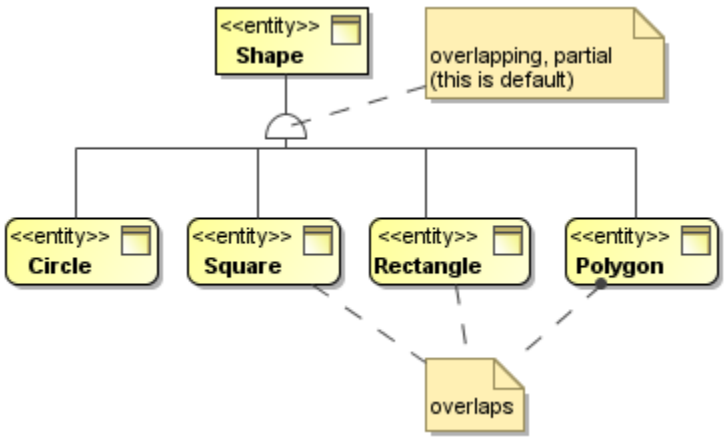
Generalizations can be joined into generalization sets (trees of generalizations), which allow specifying additional properties on a group of generalizations - such as disjointness and completeness constraints.



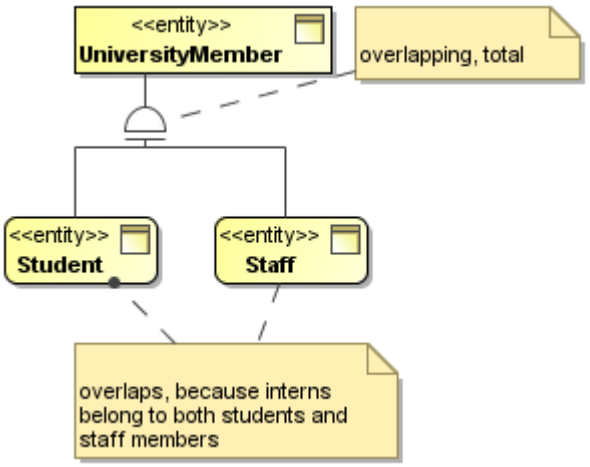

Example of generalization in ER diagram.

Disjointness and completeness constraints are specified using the **Is Disjoint** (*true* for disjoint, *false* for overlapping specialization) and **Is Covering** (*true* fo r total, *false* for partial specialization) properties. They can be set in either the relationship shortcut menu or the Specification window.

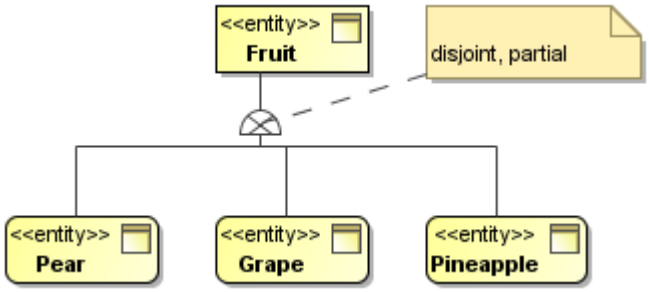
There are 4 combinations of these two settings. The “breadloaf” symbol joining generalizations into a tree shows these 4 variations (See the following figures).



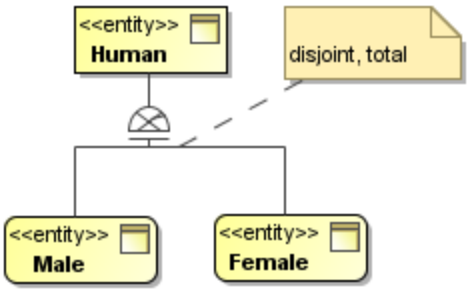

Example of overlapping and partial specialization in ER diagram.




Example of overlapping and total specialization in ER diagram.




Example of disjoint and partial specialization in ER diagram.

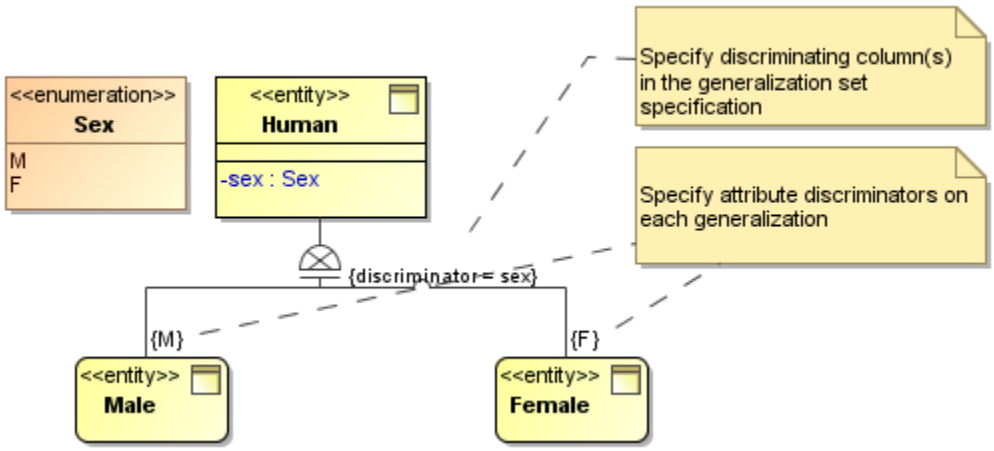



Example of disjoint and total specialization in ER diagram.

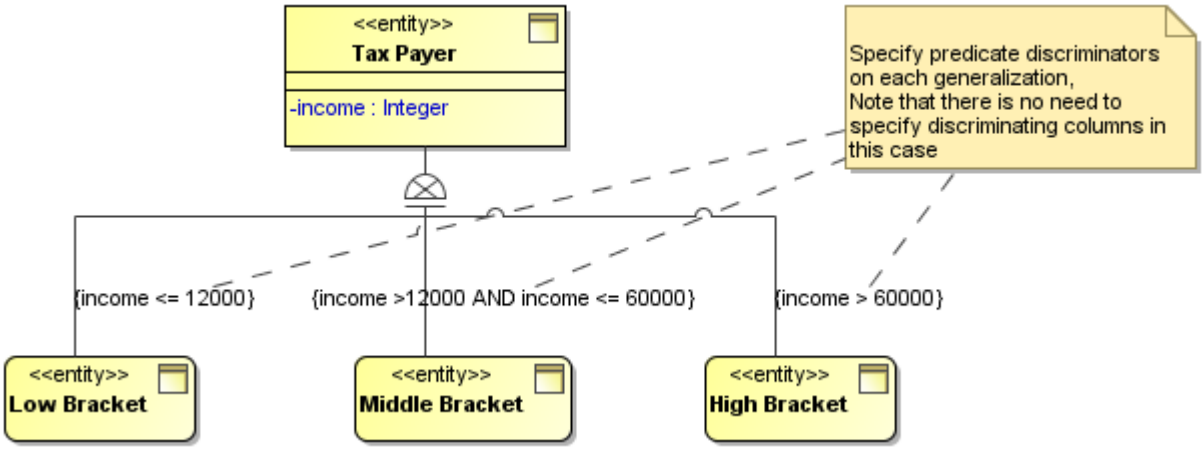
> **Note**
>
> UML terminology (covering / not covering) is used for the completeness property name in the Specification window. Other names, more familiar for data modelers, are total / partial and complete / incomplete. These terms are analogous and can be used interchangeably.

In the specialization hierarchies, there can be several ways to assign an entity instance to a specific entity subtype. You can determine it when you decide to which subtype a given instance belongs (user-defined specialization). You can also determine it by actual data values of an entity instance (attribute-defined specialization). The latter case can be further subdivided into two subcases - simple attribute-based discrimination (when discrimination is performed by doing simple attribute value comparison) and more complex predicate-based discrimination (when discrimination is specified using more complex, explicitly specified conditions).

Examples of these two cases are shown in the following figures.




Example of attribute-based discriminator in ER diagram.




Example of a predicate-based discriminator in an ER diagram.

Discriminators are modeled as special constraints placed on individual generalization relationships. The easiest way to access them is from the shortcut menu of the generalization.

The predicate-based discriminator is simpler; just fill in the **Specification** field of the predicate with an appropriate expression text.

The attribute-based discriminator is more complex. First, specify the columns by which you will sort the entities into the corresponding subclasses. This is done by filling in the **Discriminator** field of the generalization set (you can specify one or several columns there). Then, fill in the **Template** field of the predicate. This template field holds an instance specification used as a template or etalon to differentiate the entity instances into appropriate subclasses. Fill in the slots for the same columns you indicated on the generalization set.

> **Note**
>
> The category (also known as union) concept is currently not explicitly supported. Total (but not partial) categories can be “simulated” using the total specialization tree, just visually reversed.