# Modeling - Structure

Until now we’ve looked at tools to find classes and events from a deciphered problem and application domain.

This is going to be regarding how the classes and events are related to each other.

See Slide 3 for an example of an event table for last exercise modules.

## Activities in “Analysis of problem domain”

See Slide 5 to see where we are in the process.

First, we look for candidates for structure. We study 1. abstract and static relations, 2. concrete and dynamic relations between classes and objects. We identify generalizations, aggregations, associations, and clusters.

Thereafter, we explore what patterns we can apply to our structure.

Then we evaluate whether the structure is used correctly. The structure should be conceptually true. It should also be as simple as possible. This is because it is supposed to supply us with a UML diagram, which can become very messy very quickly.

## Class structures and object structures

Class: static, conceptual relations between classes. These are relations that never change. They concern all the objects of the classes.

Object: Relations that can be dynamic and changing. Some objects can be in such relations, while other objects from the same class may not.

We will look at the two relations: aggregations and associations.

## Generalization

Pretty much a “is a” relation. For example, a taxi **is a** passenger car. A maths teacher **is a** teacher. See slide 6/9 for an example of the generalization in UML and why you want to keep it simple. This is usually where you begin to think about inheritance in object oriented programming.

Note: if a class in the UML diagram is written in *italic****,*** then it is an abstract class.

See slide 10 for the vent table for the hair saloon. We readily see classes that can be generalized, like apprectince and assistant into an employee class.

## Cluster

We group classes that are conceptually tied together in what looks like a folder structure. See slide 12. Note that there is no line between owner and clerk, but they are clustered together because they are conceptually related.

## Aggregation

Usually annotates one of three alternatives:

**Whole - part**, like “the car is made up of these parts”

**Container - contents** - The main class is the container, the aggregated classes are the contents.

**Union - member** - a bit like container-contents, but it makes a bigger distinction when the contents are too few and far between.

The 1’s and 2’s in the diagram mean this: A cylinder can only be on one car, but a car can have two or more cylinders. An engine can be in one car, and a car can only have one engine.

## Association

A “knows” or “associated-with” relation.

A person can have a relationship to a car, or not have one at all. This is why the multiplicity 0..\* is at the car. A car has at least one owner, but can have more, which is why multiplicity 1..\* is at the person. Slide 14.

See slide 15 for more candidates for relationships. Customer and Appointment share the same events, so we can assume that they have a relationship in some way. The relations should be an association between the two. Furthermore, an appointment can only have one customer associated with it.

## Patterns

### Role pattern

See slide 17. A person can have roles, which is annotated by a mix of aggregation and associations. A person can have several roles, and a role can only be aggregated with one person.

### Relation pattern

The top right relation doesn’t say much about the relation between the two classes.

Instead, we create a class that annotates the relation between the two classes or objects. It’s important to note that in the example, the person aggregates (i.e. owns) the ownership, the car does not.

### Hierarchy pattern

We can have multiple layers where each layer organizes the layer below. The structure should be self explanatory, see slide 19.

### Descriptor-item

In a library, they will probably have several copies of the same book, but don’t want a unique description of all the copies, because it’s the same book. The slide 20.

## Evaluate

Once we have our candidates for relations, we need to see if they satisfy some criteria. First and foremost, the relations must be used correctly.

They must be conceptually right, names, concepts and structures should correspond to the users understanding.

The structure must be simple, avoid unnecessary generalizations and aggregations. Check against the system definition.

It’s not always obvious what kind of relation we should choose. For example, an argument could probably be made for all the structures on slide 23, but it should probably be association because of the nature of university lectures.

**When we consider relations:**

Can objects exist independently of each other? Are they equally ranked and can the connection be changed for one pair of objects to another? If you answer yes to two or three of these, then it’s probably association. Otherwise, it’s probably aggregation.

See slide 27 for the rest of the relations between the classes. See also that the “Plan” class was removed and replaced with four now classes, in effort to employ the patterns.

Note that in the last slide of the example, appointment is associated with Work because when an appointment is made, the day schedule changes because a new Work time period has been created.

When we have a diagram, we evaluate it against our event table. Do the relations between our classes in the event table correspond to our newly constructed UML diagram?

To round up: **Class structures** include **generalizations** and **clusters. Object structures include aggregations** and **associations**.