

This is one possible solution for the conceptual class diagram of exercise 1).

Classes:

Whish of the customer, flight information, ticket, payment, customer.

The customer participates in an association but we are not interested in further attributes of the customer so we use a simplified notation for him.

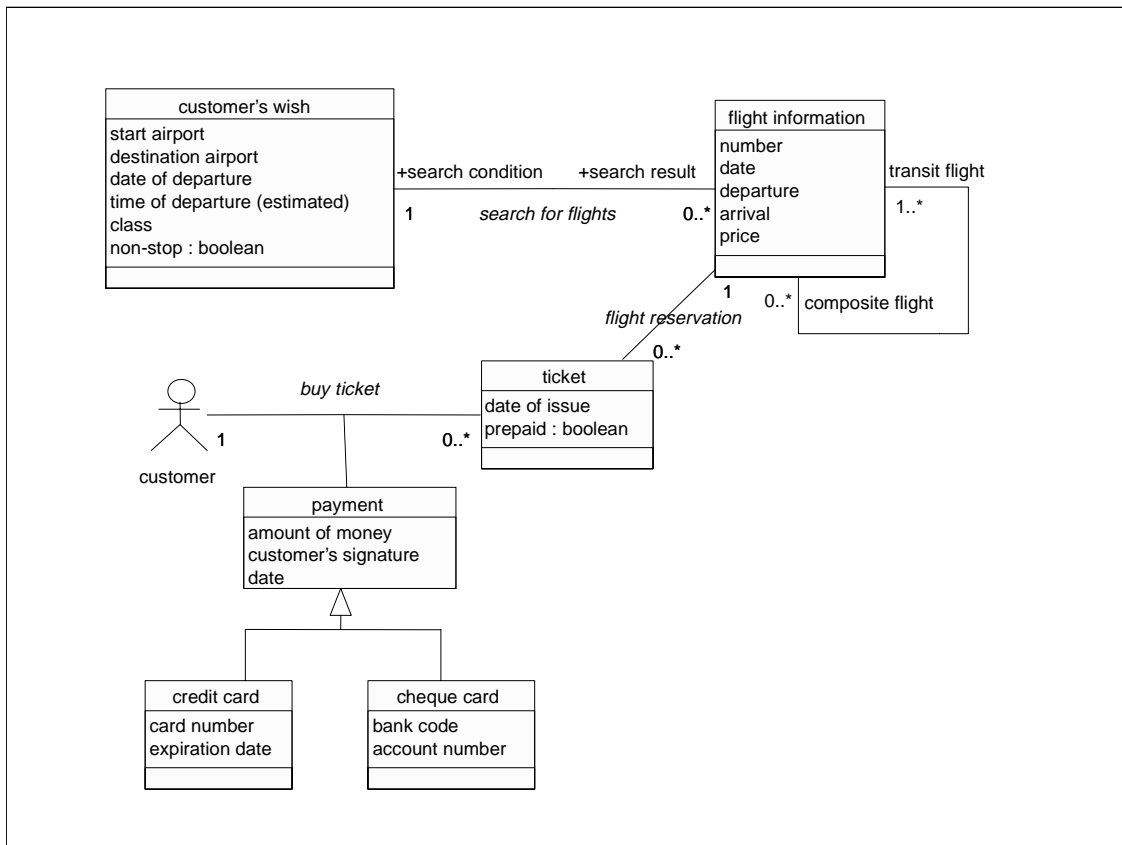
Payment includes the attributes that are common for all the supported methods of payment. Attributes that are specific for a certain method of payment can be found in the subclasses „credit card“ and „cheque card“.

Associations:

A search for flights may result in zero or more flight information objects which match the customer's wish. Each flight information object is created to respond to exactly one search condition as it is specified by the customer's wish.

Each flight information object can be used to create zero or more tickets (flight reservation). Each ticket refers to exactly one flight information.

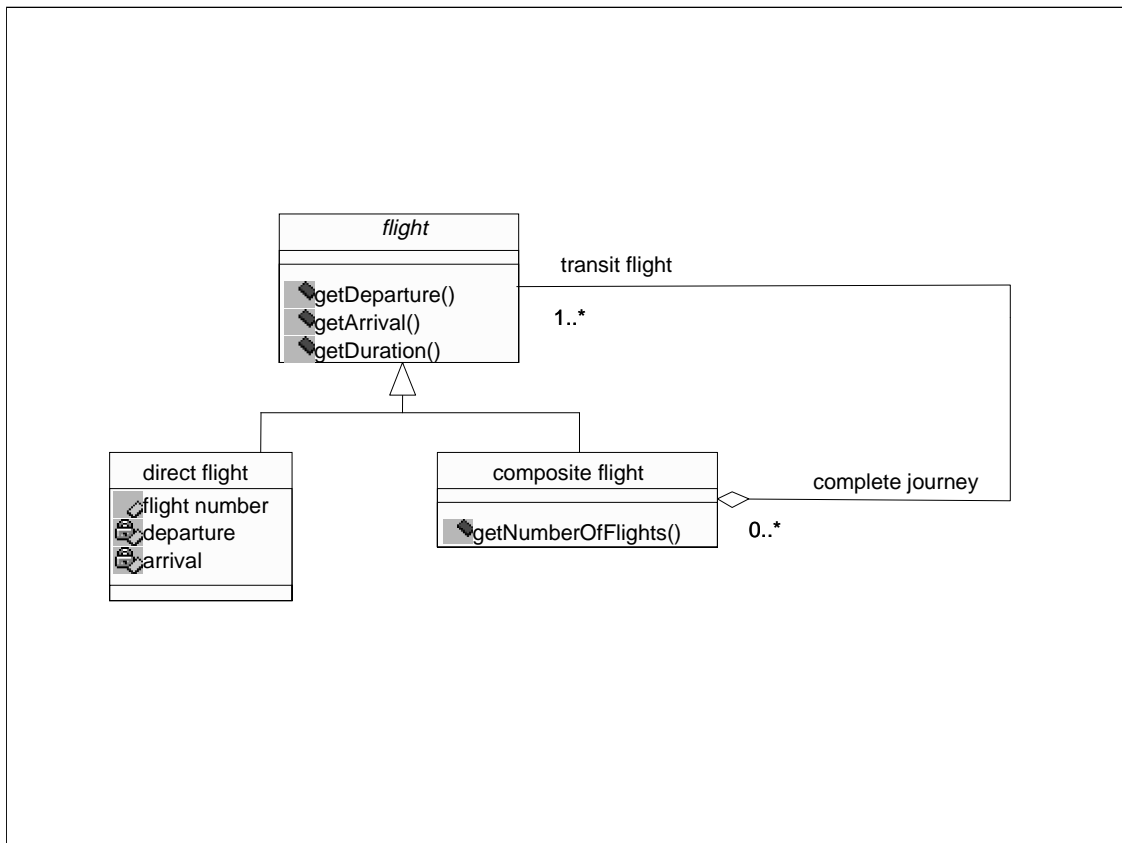
Each customer can buy zero or more tickets, but each ticket is sold to exactly one customer. The payment which is a complex property of the buying transaction is attached to this association as an association class.



This diagram is similar to the previous one but it includes a solution for exercise 2 a).

In this diagram there is a recursive association referring to the flight information class.

Each flight information may act in the *role* of a *transit flight* which can occur in zero or more composite flights. But each flight information may also act in the *role* of a *composite flight* where it consists of one or more transit flights. This model is appropriate as long as the structure of the flight information is the same for both direct flights and composite flights. If the class structure differs between the „composite flight“ and the „component flight“ another model is necessary. See next slide for details.



This is an alternative for modelling composition 2 b).

The (abstract) class *flight* models the properties that are common to both direct flights and composite flights. Common requirements are e.g. that we want to get the departure or arrival time of the flight and that we want to get the duration of the journey. As these pieces of information may require some calculation we model them as operations. How the calculation is actually done is a task of the subclasses.

Objects of the subclass *direct flight* may calculate the duration of a journey by subtracting the value of the departure attribute from the value of the arrival attribute. For *composite flights* this is not as easy.

To calculate the duration of the overall composite flight, which can consist of any number of transit flights, we have to subtract the departure time of the first flight from the arrival time of the last flight. So each composite flight aggregates a set of transit flights.

In addition we want to allow, that a composite flight can be composed of direct flights or of other composite flights again. So a composite flight just aggregates *flight* objects. For the calculation of a concrete composite flight's duration it is not necessary to know the concrete class of the transit flights, as long as we know that all concrete subclasses of *flight* implement the *getDeparture*- and *getArrival*-operation.