Summary

Comparable objects are frequently used in different applications, when using Must for simulation purposes. Therefore, such objects seem suitable for generalisation. Defining these standard-objects in units prevents the same object from being programmed more than once and, furthermore, simplifies the structure of the main-program. Object-oriented programming is very useful: it creates the opportunity to apply descendants of the generalised objects and it is possible to fit these descendants to the specific demands of the application.

Elaboration of an example in which a pay-desk and a multiple pay-desk system have been generalised gave the desired results: they indeed appeared suitable for generalisation; definitions of these objects have been assimilated in units. When initiating these objects, some parameters can be assigned, like queue disciplines and numbers of queues. As a result, the objects can be used in different applications. Besides, the generalisation leads to a relatively easy structure of the main-program. If also a customer (of which an attribute had to be accessible for the pay-desk) is defined at the same level - i.e. as a generalised customer in the same unit as the generalised pay-desk - the main-program will become even less complicated. The main-program mentioned here concerns the simulation of a simple model of a supermarket: customers are created by a generator and disappear after treatment by the right pay-desk.

So, generalisation can lead to a widely applicable object and an easy main-program. However, the demands 'widely applicable' and 'easy main-program' are often in conflict. The problem is that an object defined in a unit does not have access to attributes of an object defined at another level (e.g. the main-program). The use of virtual methods can solve this problem. This will complicate the main-program. The problem can be avoided by defining both objects in the same unit. However, things will get less flexible: it is just a solution for a specific application.