1. Introduction

Description

This book is about *object-oriented programming* as *modeling*.

Programming has to do with making *applications* (for short *apps*) on computers. Computers typically have a number of *apps* . There are numerous examples of tasks carried out by apps. This includes office tasks, like mail systems, and word processors. Other examples include large and comprehensive tasks, like reservation systems (travels, hotels), health care systems, banking systems, and control systems in trains, vehicles, and airplanes.

While other machines are typically made for a specific task, computers are unique machines in the way apps on these machines are made: They are made by creating *programs*, that are turned into apps. The creation of programs (and thus apps) is called *programming*, and it is said that computers are machines that can be *programmed* to do various tasks (of an app) and not just a specific task. The languages in which programs are made are called *programming languages*.

Object-oriented programming is the dominant style of programming and supported by mainstream languages, like C++, Java, C# and Python. Most textbooks on programming focus on the technical aspects of a programming language by explaining how the various language mechanisms work. In this book, we in addition focus on how to ensure that a program reflects the relevant elements of the application domain by describing a model of these elements.

A computer understands a very simple programming language called a *machine language*. A machine language is a very primitive language, and it is time consuming to describe programs in a machine language. For this reason, a number of *high-level languages* that are easier to use for people have been developed over the years. These languages include Cobol, FORTRAN, Algol 60, Pascal, C, SIMULA, Ada, Smalltalk, C++, Self, Java, C#, Python and many more.

In order to execute a program in a high-level language, the program must be translated into machine language for the computer to execute the program. The translation of a program written in a given programming language to machine language may be carried out by an application, which is usually called a *compiler*.

An app is actually a program in machine language, sometimes supplied with data in one form or another. For this reason, we from now on use the term ‘program’ instead of ‘app’.

When a program is *executed*, the computer generates a *computational process*. For object-oriented programming such a process consists of *objects* that hold *data–items* and carry out *activities* that may manipulate objects. During this process, there may be interactions with people and other computers. The component in the computer that may execute programs is often called a *virtual machine*, abbreviated to *VM*.

A program is a *description* of the computational processes that is generated when the program is executed. A programming language therefore includes mechanisms for describing objects, data-items, and activities. This includes *expressions* for computing values from constants and data-items, statements that describe actions to be carried out, abstractions describing classes of objects and/or classes of activities. A programming language is a formal notation and thus closer to a mathematical notation than to a natural language. The clauses in a programming language are often defined using special symbols and reserved words called *keywords*.

Modeling is an important part of programming. In order to understand what to program, the relevant aspects of the application domain has to be identified and reflected in the program.

The model as described by a program is the computational processes generated by execution of the program.

For the purpose of modelling, special modeling languages like UML may be used, combined with turning UML diagrams/descriptions into programs. Using UML or other modeling languages may be useful, but in this book you will learn how to do *modeling* <u>as part of</u> *programming*.

Using the book

Skal denne her test være en selvstændig section
1.7 Reading this book/Guidelines for reading/using this book/How to use this book?
Also the abbreviations `-"-, :::, ...` should perhaps just be introduced when used the first time?
2024-09-27: I think this section shall appear earlier, perhaps just after the front page – the reader should be told how to proceed: students, skip the preface, notation, …

We have attempted to use examples where we emphasize the domain where they belong and what the model/program is meant for.

It is important to emphasize that the design of models should always be made in close collaboration with domain experts and users of the program being developed. In addition, it is important to notice that a model is always defined for a specific purpose that guides the design of a given model.

In this book, it has not been possible to involve domain experts and users for the examples being used – they are made with the sort of common knowledge the authors may have of a given domain, and as such they may not be useful for a real program. There is no such things as a complete model that can be used for all purposes.

All of the examples are thus what may be called toy examples.
They can all be compiled and executed by the qBeta compiler except for ….

The chapters and sections of the book need not be read sequentially – some parts especially some of the examples may be skipped during a first reading – +++ do we here says which chapters/sections must be read sequentially?

Abbreviations

For practical purposes, we do not always show all the code in the examples and we use the following notation of code not shown:

- The symbols `-"-` stands for code, which has been shown in a previous example.
- The symbols `:::` stands for code, which will be shown later – often in the same section.
- The symbols `...` stands for code, which is not shown and left to the reader to fill in.