Different Views On Object-Orientation

Sample IA for IOOR, 2006
15th November 2006

Task

Read the 1991 revised version of [Stroustrup1988] (i.e. [Stroustrup1991]) with a critical mind. Also read [Smith1994]. Then write a short paper (~4 pages or 1500 words) on how/if the views on OO differ in these papers.

Introduction

In this assignment I am going to compare the view on Object-Orientation as it is given in the 1991 revised version of Bjarne Stroustrup’s article “What Is Object-Oriented Programming?” [Stroustrup1991] with the views on Object-Orientation of Mark Lentczner, Walter R. Smith, Antero Taivalsaari and David Ungar given in Randal B. Smith’s article “prototype based languages (panel): object lessons from class-free programming” [Smith1994].

I will start with a short overview of the two articles followed by the comparison. Finally I will draw some conclusions.

Stroustrup wrote his article because he saw a need for definitions for the terms that are used in the context of object-oriented programming. “Object-Orientation” was a buzzword at that time and lots of programming languages suddenly claimed to be object-oriented or at least were extended with so called object-oriented features. Stroustrup wanted to define (even though just “informally”) what object-orientation really is. He attempts that by first telling the difference between data abstraction (as it was found in many programming languages) and object-oriented programming. Then he presents what features a programming language must have to allow programming object-oriented style.

Smith’s article is of a totally different nature. It is the result of a panel debate and presents the views of four developers of prototype based, class-free languages. They give an overview and point out the benefits of “their” languages. Finally they demonstrate how to solve a simple exercise in their language.

Comparison of the Different Views

The comparison of the views on object-oriented programming will be done dialectically, i.e. one opinion of one of the authors will be presented and then compared with the corresponding view of the other author. After that the next
point will be picked out. The benefit of this kind of analysis is that a more direct comparison is possible and the difference of the views of the authors are pointed out more clearly.

Stroustrup defines the paradigm of object-oriented programming as a superset of the paradigm of data abstraction. According to him the additional principle, which distinguishes object-orientation from data abstraction, is the possibility to create a data type which is a specialisation of another data type\(^1\) (pages 6, 7) i.e. inheritance and type hierarchies.

In Smith's article inheritance also plays a big role as each of the four presented programming languages either has the feature of inheritance or something similar. Nevertheless there is a big difference between the inheritance described by Stroustrup and the inheritance in the four prototype based languages. Stroustrup uses inheritance for substitutability whereas in Smith it is described as an “organizing tool for the programmer, not as a pure abstraction mechanism” (page 104). Inheritance is used to have the “same information in two places at the same time” (page 106). Type hierarchies are not given at all in the object-oriented languages of Smith's article since types in the common sense do not exist in the described languages.

Stroustrup states that the programming paradigm of object-orientation is to “Decide which classes you want; provide a full set of operations for each class; [...]]” (page 7). With this statement he introduces the fundamental brick of object-oriented programming, as it seen by him: the class.

Classes as they are described by Stroustrup do not exist in the class-free programming languages described in Smith's article. Here objects (or rather prototypes) are the fundamental construct. Classes as blueprints for object instantiation are not needed because objects are created directly by the programmer and then just instantiated or copied to have several “instances” or “peers”. Lentczner for example states in his description of the language *Glyphic Script* that they left out classes because they would “still represent a level of abstraction away from the elements of the program” (page 104). Similar claims are made by Ungar (page 107). Nevertheless some of the presented languages have classes, but you can (and also should) program in them in an object-oriented way without using classes at all.

According to Stroustrup inheritance and method look-up / virtual functions / overriding of methods are another important part of object-oriented programming languages. He concludes that a language which supports\(^2\) object-orientation, needs both of these features (page 16).

The languages discussed in Smith's article support these two features in a slightly different way: The differences in inheritance were already discussed above. The difference concerning methods is that methods and attributes (variables) are accessed using the same mechanism in most of the presented prototype based languages (page 105). The important features (according to Stroustrup) however are given in these languages, too.

Stroustrup says that the language should support the programmer (page 14). He thinks of powerful abstraction mechanisms and a secure type system which takes care of possible mistakes a programmer could do. He mentions that this

---

\(^1\)\ e.g. a circle or a rectangle is a specialised kind of a shape

\(^2\)\ i.e. has “facilities that make it convenient to use that style” (page 2)
is perhaps not the most flexible way one could take but for him this is more important.

Smith et al. on the other hand are convinced that prototype based programming is “much more direct” (page 109), “cognitive more lightweight” and “more naturally” (page 111) apart from that they also claim that prototype languages are more straightforward and more beautiful in design (page 111).

For Stroustrup the optimal object-oriented programming language should be flexible in that way that it is a general purpose language. He notes that it should be useable on all kind of machines, that it should be able to interface with operating systems, that it should be fast and efficient enough and that it should be useable in all field of applications (page 19).

The prototype based languages still have some efficiency problems and are sometimes not designed to work with all kind architectures and operating systems (e.g. Newton Script, page 109). But nevertheless they don’t seem to ignore efficiency issues. In particular they try to optimise the amount of memory objects need (page 109) and are in general considered to be “robust and efficient enough to support real world applications” (page 102).

Conclusions

All in all Stroustrup’s view on object-oriented programming and object-oriented programming languages differs a lot from the view(s) of Smith and his fellow authors. The major difference between them is more a “philosophical” one. For Stroustrup it is important to be able to build abstract models with strict type hierarchies that prevent the programmers from making mistakes through static typing. In contrast to that the proponents of the prototype based languages claim that their approach is much easier for the programmer to work with since one abstraction layer (classes) is left out.

Stroustrup’s definition of object-orientation seems very near to the design of C++. According to him everything which is needed in and object-oriented programming language is present in C++ and features other object-oriented languages have are declared as minor or even harmful or unnecessary. As an example he declares the lack of garbage collection as a feature and refers to the finalize method which lets the programmer do the same, but more efficient. I think that these are some caveats in his “philosophy”. On the one hand he wants to protect the programmers against errors which could evolve through dynamic typing and on the other hand he lets the programmers do critical things themselves (e.g. freeing of allocated memory).

Smith et al.’s view seems to be more academic (although none of the panellists is actually from an academic institution). They know that their field of research is rather young and point to questions without actually having an answer. When discussing Newton Script, it is mentioned to be a proof that prototype based languages are useable in the real world as this language is being used in an actual product. I think that the view of Smith et al. is a bit aloof. They claim that their concepts are easier for the programmer—without having real evidence (the reader just has to rely on what they state).

In conclusion I think that Stroustrup’s view is based to much on C++ (which is quite natural). I slightly prefer the view of Smith et al., but in general I think that there are some good points in both articles. So a mixture of both points
of view is perhaps the right way to go (as usual?!).

References


In this assignment the 1991 revised version of this paper is discussed.