Object orientation in the literature and in education

Abstract: The efforts around the world – CS4All in the U.S. or Computing At School in Great Britain – show that computing literacy is seen as important. One important part of computer science education deals with learning programming. So, object orientation should be in focus. But what is object orientation? Several different definitions are presented, and a definition of object orientation by its fundamental concepts is introduced. Furthermore, several educational “paradigms” are discussed. Additionally, a choice of object-oriented programming languages is presented. After all that theoretical background, some exemplary implementations of object orientation in national (German) and international curricula are shown. All in all, the article provides a broad overview of the topic of object-oriented programming in computer science education.

Keywords: Object orientation, object-oriented programming, computer science education

ACM CCS: Social and professional topics → Professional topics → Computer science education, Social and professional topics → Professional topics → K-12 education, Applied computing → Education

1 Introduction

The introduction of Computing At School (CAS) in Great Britain and CS4All in the U.S. are two recent cases where computer science is introduced as a mandatory subject. Although there is a consensus that digital literacy is necessary, policymakers are still struggling with implementing this need in an independent subject in secondary education. In contrast to this ongoing discussion in primary and secondary education, the role of object-orientation for programming can be seen as consolidated. The unique situation in Germany with its federally organized educational system resulting in states that have a mandatory subject “Computer Science” at the lower secondary level and states that have not, condenses all these discussions. To enable communication between policymakers, teachers, and researchers in a common language, it is important to clearly define the underlying concepts, teaching approaches, and existing curricula.

This article clarifies what object orientation is and how it is taught. For that reason, an overview of different definitions and a synthesis of the evidently investigated most common concepts are conducted. Afterward, educational approaches on the order of introducing the basic programming concepts (i.e., concepts belonging to either the imperative-procedural or the object-oriented paradigm) are presented. The last part of this paper gives several examples on how object-oriented programming is realized in different curricula (e.g., national and international secondary school curricula, ACM computer science curriculum). Here, the focus is on the implementation of object-oriented programming in general and not on single object-oriented concepts.

2 Object orientation

Many different definitions or specifications of object orientation can be found in the literature. Unfortunately, it turned out that there are several substantially dissent definitions of object orientation that cannot be integrated into one “mainstream” perception.

In the following, some prominent views of object orientation are summarized. Starting with the early definitions by Dahl and Nygaard, several examples from different sources are presented here to demonstrate the variety.

2.1 Definitions

Dahl and Nygaard established the notions of classes and objects in the context of the development of Simula [15]. They stated that the “class concept introduced is a remodeling of the record class concept proposed by Hoare. [...] A prefix notation is introduced to define subclasses organized in a hierarchical tree structure. The members of a class are called objects. Objects belonging to the same class have similar data structures” [16, p. 159]. Additionally, they introduced the notion of inheritance on a class
level and the combination of data and objects that belong to a class.

Bertrand Meyer pointed out an important difference between the object-oriented paradigms and other programming styles. While the latter regard data as passive, the object-oriented paradigm focuses on active objects that operate on their own data [38].

Later, Meyer defined seven concepts that have to be implemented to fulfill the requirements of object-oriented paradigm: object-based modular structures, data abstraction, automatic memory management, classes, inheritance, polymorphism and dynamic binding, and multiple and repeated inheritance [40]. This definition is comprehensive, although it is more a definition of object-oriented languages than of object orientation in general.

Another approach for defining object orientation was conducted in [51]. Wegner’s definition is built hierarchically from the concept of an object, adding the concepts of class and inheritance and is based on the classification of programming languages. The first group of languages is object-based. By adding the concept class, the remaining languages are called class based. The last and most restrictive language group includes the object-oriented languages with the introduction of inheritance.

A definition that is neither focused on languages nor the object, in particular, was the one by Blair [9]. He defined four dimensions of object-oriented systems, not only languages, which were all based on object-oriented computing as a particular kind of abstraction: Encapsulation, Classification, Polymorphism, and Interpretation.

In the early 1990s, Rumbaugh, Jacobson, and Booch introduced object-oriented modeling techniques. Later on, these different approaches were integrated into the unified modeling language (UML). The first version was published in 1997. In the context of UML the following view of object orientation was formulated:

“In [the object-oriented] approach, the main building block of all software systems is the object or class. Simply put, an object is a thing, generally drawn from the vocabulary of the problem space or the solution space; a class is a description of a set of common objects. Every object has identity (you can name it or otherwise distinguish it from other objects), state (there’s generally some data associated with it), and behavior (you can do things to the object, and it can do things to other objects, as well).” [10]

2.2 A definition of object orientation by its fundamental concepts

It seems natural to solve this problem by accurately defining the most important concepts of object orientation. But how find these most important concepts? For this purpose, Deborah Armstrong [4] investigated many sources of literature. She searched for the keyword “object-oriented development” and found 239 scientific articles. 88 of those asserting that a particular set of concepts characterized the object-oriented approach. These 88 distinct sets were evaluated, counting the relative frequencies of the addressed concepts.

It turned out that eight of these concepts were mentioned in more than 50% of the sources: Inheritance, Object, Class, Encapsulation, Method, Message Passing, Polymorphism, and Abstraction. Honoring this particular importance, she called these eight concepts the “quarks of object-oriented development” [4].

The next subsections describe and define these eight “quarks”. Please note that this is done in logical order, not according to the order of Armstrong, starting with the name-giving concept “object”.

2.2.1 Object

The name-giving concept of object orientation was introduced in Simula67. According to this, the idea of the concept object is related to a representation of the real world [47]. The dualism of object as a data carrier and something that executes actions [4] is also pointed out by Blair: “An object is an encapsulation of a set of operations or methods which can be invoked externally and of a state which remembers the effect of the methods” [9, p. 26]. More precisely, the state of an object is defined by the values of its attributes [13].

2.2.2 Class

According to Booch et al., “a class is a description of a set of common objects” [10]. All instances of a class share the same kind of data, even if they differ in their state (data values). In turn, according to Blair “a class is a template from which objects may be created. It contains a definition of the state descriptors and methods for the object” [9, p. 29]. Additionally, in many definitions the concept of class is also used as a data type for the corresponding object [13]. Some definitions even emphasize class as the most important concept in the object-oriented approach [27].

2.2.3 Method

The implementation of the behavior of an object is called a method. Nevertheless, a method is nothing but a func-
tion strictly bound to an object. According to Armstrong a **method** is “a way to access, set or manipulate an object’s information” [4], while according to the definition of Blair, **methods** are the only possibility to change the state of an object [9].

### 2.2.4 Message passing

As mentioned above, the activity and communication of and between objects are one of the fundamental ideas of object orientation. This communication is enabled by the mechanisms of **message passing**. According to Armstrong **message passing** is “the process by which an object sends data to another object or asks the other object to invoke a method” [4, p. 126]. Rosson et al. emphasize the communication aspect of **message passing** by pointing out the communication between objects by the ability to send and respond to messages [43]. This point was clarified by Sethi. A “message to an object corresponds to a procedure call; messages can carry parameters. In response to a message, the object executes a method, which corresponds to a procedure body; it returns an optional result. The result, if any, will itself be an object of some class” [47, p. 258].

### 2.2.5 Encapsulation

In the literature, **encapsulation** is defined as a collection of data that is only accessible through well-defined processes. Additionally, there are definitions only focusing on the grouping of data and the corresponding functionality [20] and those that emphasize the well-defined accessibility of that data [27]. The balance between readability of code and the **encapsulation** of data in single classes is a central topic of this concept [22].

### 2.2.6 Polymorphism

Blair postulates **polymorphism** to be “one of the most characteristic features of object-oriented systems” [9, p. 35]. According to Armstrong, “polymorphism is defined as: the ability of different classes to respond to the same message and each implement the method appropriately” [4, p. 126]. Cardelli et al. listed different aspects of **polymorphism** [12]. They distinguish between universal and ad-hoc polymorphism. Universal polymorphism, as well as ad-hoc polymorphism, is again differentiated into two types. The first type is parametric and inclusion polymorphism; the second type is overloading and coercion.

### 2.2.7 Inheritance

The concept of **inheritance** plays a central role in most definitions of object orientation. It was introduced with the development of Simula67 [16]. According to Armstrong **inheritance** is “a mechanism that allows the data and behavior of one class ([superclass]) to be included in or used as the basis for another class ([subclass])” [4]. In the subclass, the original attributes and methods can be changed by overriding [46]. By this way, **inheritance** supports code reuse on the one hand, and on the contrary, provides the feature to build class hierarchies. Additionally, together with polymorphism, overriding builds the powerful basis of object-orientation [39].

A particular challenge in this context is multiple inheritance. If a class is derived from more than one class and at least two superclasses provide the same data or functionality, there must be a mechanism to uniquely define which data or which functionality can be used by the subclass. Some languages provide multiple inheritance such as C++, while others, such as Java, do not. Whether multiple inheritance is used in a language is a choice between the advantages and disadvantages and depends on the field in which the language is mainly used [46].

### 2.2.8 Abstraction

The central concept of **abstraction** is “the act of creating classes to simplify aspects of reality using distinctions inherent to the problem” [4]. By this way, object orientation enables us to model parts of our world naturally. Consequently, this real-world modeling approach was one of the central aspects that led to the paradigm shift from imperative procedural programming to object-oriented programming [42].

### 3 Educational approaches to object orientation

After defining object orientation, different educational approaches to object orientation and object-oriented programming, in particular, are presented. First, an investigation on the role of programming in computing education
is presented [45]. Schulte summarized educational characteristics of programming and discussed programming and the goals of education. He stated that automation is a fundamental requirement of programming. Furthermore, according to Schulte, programming is a particular type of interaction with the computer using an abstract notation. Besides the fact that programming includes coping with the complexity, it includes the creative process of seeking and finding solutions. The following two sections present different possible sequences of introducing object orientation and object-oriented programming and a summary of different possible programming languages and their suitability for education.

3.1 A suitable educational “Paradigm” for introducing object orientation

According to the significant paradigm shift from procedural to object-oriented programming, the sequences are called “educational paradigms”. As van Roy et al. mention, object orientation is sometimes regarded as difficult and so, hard to teach. According to them this “might be true if you teach it to students who have a background in another programming paradigm” [50, p. 270]

Many sources are describing the methodology of introducing the object-oriented paradigm in an introductory course. Most of them comprise the approaches of “objects-first” or “objects-later”. Besides the simple approaches, there are other methodologies for introducing programming, which affects the different aspects of object orientation in general (modeling or programming) and the time of introducing specific programming concepts in particular. Figure 1 gives an overview of all these approaches. Modeling includes the introduction of all modeling aspects. Object or Class imply the introduction of the concepts of object and class, whereas, Object-oriented programming includes the programming notions of the object-oriented paradigm. Finally, Algorithm/Control structures includes the notions of algorithms and their implementation including control structures. The following paragraphs introduce different studies on the presented approaches.

In 2008, Bennedsen et al. [6] asked over 700 teachers about their understanding of objects-first. Their answers are categorized into three categories: using objects, creating classes, and concepts. While the first two types are clear concerning their meaning, the last one deals with the focusing on the object-oriented model creation to cope with the conceptual aspects of object orientation.

A synonym for the objects-first approach is the notion of starting with an object-oriented design. Here, the modeling of objects is the first step. Furthermore, concepts like inheritance, polymorphism, and generalization are introduced by design [3].

Diethelm [21] introduced a stricter version of the objects-first approach. She states that most problems with object orientation – and especially the modeling aspect – lie in the use of classes instead of objects. This approach is called strictly-objects-first. Another issue mentioned by her is an approach that applies modeling techniques for introducing the object-oriented notions. This approach is called models-first. Based on this idea, objects are first modeled on their own and then classified. Both modeling and programming should focus on objects rather than classes.

The objects-later methodology for learning object orientation also emphasizes the modeling aspect. In contrast to the objects-first approach, it starts with algorithmic concepts including control structures and introduces object modeling after that. The problems and effects of the objects-later approach have been discussed in the literature, particularly in the beginning of the change to the object-oriented paradigm; for example, see [19].

In several publications, the objects-later approach – with the paradigm shift from procedural to the object-oriented paradigm – is said to be very challenging and not applicable in introductory courses [44].

Most courses have now changed to an objects-first approach, which is either more or less strict. Nevertheless, there were and are several reservations in the object-oriented approach. At the beginning of the paradigm shift
in education, Decker [18] listed the top ten reservations and attempted to discuss whether the restrictions are real or just fear of new things. In the end, they recommend an objects-first approach, even though it suffers reservations [18]. In a study published ten years later, an indirect comparison of the two main approaches described above was conducted. Decker [17] investigated the performance of two student groups in an object-oriented CS2 course. The first group had an introductory course according to the objects-first approach, while the second group started with the procedural paradigm in the very beginning (objects-later). The group that used objects from the beginning performed better in the more advanced object-oriented concepts, although the second group was also introduced to the object-oriented concepts. Furthermore, the algorithmic performance was equal in both groups, which suggests that both approaches are successful in the procedural topics [17].

3.2 An appropriate language for an introductory programming course

Knudsen et al. [34] focus on the fact that learning to program in an object-oriented manner is more than learning a new language that is object-oriented. Nevertheless, in the last few years, a large variety of languages have been developed. According to Pears et al. [41] these languages can be separated into three sets. The first set is taken from industry, and the selection is substantiated with the necessity to prepare students for industries’ needs. The second group provides either a subset of a typical industrial language or an environment for one of those languages that are specially designed for educational purposes. Another set of languages is developed only for educational reasons. Most of those languages provide a graphical programming environment that is often related to a kind of gaming.

The category of languages with industrial roots are historically linked to the development of the programming paradigm. Lewis [37] presents in his paper on the myth about object orientation two widely spread languages in introductory courses. The first language is C++, which is a hybrid language. Hybrid language means that there is no need to program in an object-oriented way. The other language is Java. Even if the design of Java tends to be object-oriented, it is not a pure object-oriented language. “The existence of primitive types is one reason. [...] More important, the requirement that all methods be defined as part of a class does not automatically lead to an object-oriented design. Procedural abstraction could be allowed to dominate, destroying the conceptual elegance of a proper design” [37, p. 248].

Furthermore, there are languages designed primarily for the object-oriented programming paradigm, but unlike C++, Java or recently # they have not found broad application. A famous candidate, which has its role in the industry, but was designed with education in mind, is Smalltalk of Allan Kay [52]. One recent example is Grace, which was developed especially for novice programmers and is purely object oriented [8].

Another set of languages includes those that are related to one of the above, but mostly only with a subset of concepts. Bluej is based on Java but omits the main-method concept. In contrast, all objects are either constructed by another object or by the user in the development environment [36]. Another improvement is Stride [35] that uses frame-based editing and omits several of Java’s syntactical difficulties like semi-colons.

In the set of languages that are designed for educational purposes, some languages are only applied in a few courses and others that are used around the world. Nearly all examples have in common that they provide a kind of setting, like a game or movie, where objects can interact with each other or with the environment. The capabilities of interaction and the kind of objects vary sharply through the different languages. Examples are Alice [14] or Snap! [24].

4 Object orientation in national and international curricula and standards

After defining object orientation and reviewing educational approaches and programming languages, some examples for the implementation of object orientation in national and international curricula should be presented. The international ones are the ACM/IEEE Joint Task Force Computer-Science Curriculum [2] and the Educational Standards of the Computer Science Teacher Association [49]. The national, German, representatives are the general guidelines for the Abitur in computer science (Einheitliche Prüfungsanforderungen (EPA) Informatik [48]) and the standards of the German Society for Computer Science. Furthermore, two national curricula for computer science in secondary education are analyzed. First, the Bavarian curriculum [5] was chosen because there is a compulsory subject in computer science. Second, the curriculum of North Rhine-Westphalia [1] was chosen because...
it is one of the most current curricula. A detailed analysis of the object-oriented coverage of the presented standards and curricula can be found in [7].

4.1 The ACM/IEEE joint task force computer-science curriculum

18 knowledge areas form the knowledge body of the curriculum. These knowledge areas are not supposed to develop particular courses in the school programs. More precisely, most classes will pick topics from several knowledge areas. To provide advice on which topics should be covered by all students and which should be reserved for specialized courses, the topics are identified by a particular level. They are either “core” or “elective” and if they are core elements they can be “tier1” or “tier2”. If they are tier1 core, they should be covered by all students. Tier2 core should be covered by almost every student and additionally the elective elements should be provided, but do not have to be covered by all students. Nevertheless, only covering the core elements is not sufficient for an undergraduate degree. Besides the knowledge areas, a sub categorization is conducted. The knowledge units provide topics and learning outcomes, which have certain mastery levels that are borrowed from Bloom’s taxonomy. These levels are familiarity, usage, and assessment [2].

The curriculum covers nearly all facets of object orientation. It is even a knowledge unit within the programming languages knowledge area.

4.2 Educational standards of the computer science teachers association (CSTA)

An international approach to defining a standard for computer science education is provided by the Computer Science Teachers Association (CSTA), which is associated with the Association for Computing Machinery (ACM) [49]. Over a couple of years, a set of competencies was defined that should be included in the teaching of computer science.

The standards are organized in two dimensions. The first dimension distinguishes the students based on their age level. Three different age levels are defined. The first level starts with elementary school students who should be introduced to fundamental concepts. Grades six to nine are grouped in level 2. They “begin using computational thinking as a problem-solving tool” and “begin to appreciate the ubiquity of computing and the ways in which computer science facilitates communication and collaboration” [49, p. 8]. The third level (grades 9 to 12) focuses on the application of concepts and the creation of real-world solutions.

The second dimension defines content areas that are called strands. In the standards five strands are mentioned: computational thinking, collaboration, computing practice, computers and communication devices, and community, global, and ethical impacts.

There are observable objectives formulated as a competency facet for every strand and level.

The standards do not scope object orientation in a specific way. Only parts of computational thinking can be related to some concepts of object orientation.

4.3 Educational standards of the German Society for Computer Science (GI)

The educational standards for computer science in secondary education were published in 2008 [25] and 2016 [26] by the German Society for Computer Science. The standards were developed to ensure up to date and accurate education of computer science in schools, and to address computer-science teachers, as well as administrative deciders and teachers’ instructors. Development of the standards is based on national and international educational standards in other subjects. Additionally, general teaching principles are taken into account. Originally, the standards were published in German. If not noted, the English translations are taken from [11].

The GI standards are divided into two main parts. The first part covers the concepts, while the second part includes processes applicable to computer science. The content section itself is again divided into five sections: information and data, algorithms, languages and automata, informatics systems, and finally informatics, man and society. The processes are also divided into five sections: model and implement, reason and evaluate, structure and interrelate, communicate and cooperate, and represent and interpret. The part for upper secondary education [26] additionally contains a third dimension. There are three request areas defined: reproduction, reorganization and transfer, and reflection and problem-solving.

Besides the formulation of the observable learning objectives, in the last part of the GI standards, a variety of examples for each section are given. Additionally, the intentions underlying the objectives are explained in a detailed way.

In total, the focus of object orientation in these standards lies on the modeling aspects. Implementation of
these models is excluded in the text. All programming aspects in the standards are related to algorithm implementation.

4.4 General assessment guidelines (EPA) in computer science

The general assessment guideline (Einheitliche Prüfungsanforderungen (EPA) Informatik) for the A-levels was conducted by the German Kultusministerkonferenz (KMK) in 1989 and revised in 2004 [48]. The EPA consists of two parts. The first part comprises the written and oral exams and also contains a brief description of the observable learning objectives that form the basis for the exams. The second part provides several task examples.

The only object-oriented observable learning objectives are related to the modeling aspects. Nevertheless, there are competencies such as the ability to implement graphical user interfaces that focus on object-oriented programming. The provided task examples clarify the needed concepts. Again, the basic concepts of object, class, attribute, and method are emphasized. A concrete definition of object-oriented programming competencies is missing.

4.5 Curricula of German grammar schools

In addition to the educational documents previously presented, examples of national secondary school curricula are presented.

4.5.1 North Rhine-Westphalia

The curriculum for secondary education in North Rhine-Westphalia (NRW) is one of the most current. It is oriented on competencies. In contrast to the Bavarian curriculum described in the next section, computer science in NRW is not a mandatory subject, but an elective subject in secondary education. The curriculum defines an overall subject competency that can be divided into two parts. First, there are competency areas that contain the basic dimensions of the processes in the subject. Furthermore, there are areas of knowledge that define the basic concepts. These two facets lead to the competency expectations that are based on observable applications.

The curriculum contains two competency levels. The first level is called the introduction stage (Einführungsphase). After gained competencies have been developed, the students enter a new level, the qualification stage (Qualifikationsphase), where the former competencies should be improved and new ones gathered. The knowledge and competency areas stay the same.

In the introduction stage only the knowledge area “data and their structure” contains the basic concepts of object orientation. Similar to all of the other documents investigated, the modeling aspects are the focus. Nevertheless, implementation of classes in a programming language (e.g., Java) is a competency that students should gather. Additionally, they should be able to use predefined documented class libraries.

Within the qualification stage, object orientation is only represented in the knowledge area “data and their structure”. Similar to the introductory stage, the focus is on modeling aspects.

4.5.2 Bavaria

In 2004 the Bavarian state introduced computer science as a subject for all grammar school students in grades 6 and 7 and for students in grades 9 and 10 who chose the science and technology track. The courses in computer science are elective courses for students in the final grades [33]. The courses and the underlying curriculum are based on object-oriented modeling.

“From the point of view of general education it seems that among all themes of informatics it is object-oriented modelling that promises the most benefit for the students. Thus we chose it as the central theme of our course.” [28, p. 2]

This concentration on object-oriented modeling aspects leads to a partitioning of the learning process into three steps. In the first two grades, the general concepts of object orientation and especially of the modeling aspects are introduced. Therefore, the students model standard software documents in an object-oriented manner. In the middle two grades, students concentrate on modeling the real world by using database systems and others. Finally, the students should apply object-oriented programming to simulate their models. The final two grades are for specialization as they are only elective and address students with a broad interest in computer science. The implementation of this methodology can be seen in the corresponding textbooks; for example, [23, 29, 30, 31, 32].

The topics of the curriculum cover the basic concepts of object orientation. Besides conceptual knowledge, application of the concepts to examples from the context of the students is the main focus of the curriculum.
of grades 9 to 11, the students apply their knowledge to a bigger project, mostly in teamwork [33].

5 Conclusion

Object orientation is a buzz-word in computer science education. A search for the term “object orientation” in the ACM Digital Library leads to over 57000 results. But as shown above, there is hardly any standard definition of this term. The presented examples illustrate the broad varieties of attempts. To decide which aspects can be integrated into a curriculum these definitions can be used, if the focus is appropriate. For wider use, a definition by its common concepts seems more suitable, especially for educational purposes. The order of introducing programming concepts is a widely discussed topic. The presented approaches show that it highly depends on the purpose of the course when to introduce which concept. If the primary focus is on modeling, a models-first approach is a perfect choice. Otherwise, if the focus is on algorithmic programming, object orientation can be introduced later. But, if object orientation is in focus it should be introduced first to avoid difficulties caused by the “paradigm-shift”. The presented implementations of computer science in curricula show that there are working examples. Now, it is on the policymakers to introduce computer science and digital literacy to everyone.

References


49. The CSTA Standards Task Force. CSTA K-12 Computer Science Standards.


52. A. Wren. Relationships for object-oriented programming languages.

Bionotes

Marc Berges
Technical University of Munich, School of Education, Arcisstr. 21, D-80333 München, Germany
berges@tum.de

Marc Berges is senior researcher at the professorship for computer science education at the Technical University of Munich. He finished his doctoral in 2015 on object-oriented programming in computer science education. His current research focuses on teacher competences in programming education.