Teaching Object-Oriented Programming with the Objects-first Approach: An Experience Report

Sandro Speth[1]

Abstract: Lecturers worldwide continue to discuss suitable approaches to teach students programming concepts in general and object-oriented programming in particular. When using the so-called objects-first approach – in contrast to the established bottom-up approach – students start right from the beginning with objects and their interfaces and then only gradually learn more object-internal concepts, such as implementing operation bodies by using variables and control flow constructs. This gives students, from the start, a better object-oriented way of thinking, which is often much harder to learn properly in bottom-up approaches. In this paper, we describe the structure and design considerations and report our experience in implementing our introductory programming course based on the objects-first approach. We outline which concepts we teach, when and to which extent, and discuss the pros and cons of our implementation, considering student feedback from teaching evaluations.

Keywords: Programming Introductory Course; Objects-first; Teaching Programming

1 Introduction

Over the last decades, lecturers have regularly discussed different approaches to how students can best learn object-oriented programming concepts. The most established approaches are the bottom-up method, where students traditionally learn procedural programming first and then get into object-oriented (OO) concepts, and the objects-first approach [Me09]. In objects-first, often called outside-in, students are initially given only an API with a limited number of features provided by, for example, a mini programming world (MPW). This allows a significant amount of programming language complexity to be abstracted away from students. Typical examples of such MPWs are Kara the ladybug[2] or the hamster simulator [BB04]. At first, students only work by calling the API of a handful of objects, gradually being taught more and more concepts and thus opening up more elements of the MPW and being able to implement them themselves. This is to help students learn and practice library (re-)use and OO concepts and ways of thinking from the very beginning before going into deeper concepts such as control flow. While objects-first, in comparison to the bottom-up approach, often yields good results for students with no prior experience, students who have programmed previously criticize the approach and claim to be more bored in the early part of the lecture. Additionally, instructors often debate how much content can and should be covered in such foundational programming lectures.

[1] University of Stuttgart, Institute of Software Engineering, Universitätsstraße 38, 70569 Stuttgart, Germany
sandro.speth@iste.uni-stuttgart.de

[2] `https://www.swisseduc.ch/informatik/karatojava`

doi:10.18420/seuh2023_04

For this reason, we describe in this paper our experiences with the objects-first approach in our programming introductory course at the University of Stuttgart. We describe the content we cover in our lecture, its depth and sequence. For our course, we implemented an OO variant[3] of the hamster simulator in Java which we explain briefly. Finally, we discuss the presented experience, considering the students' evaluation at the end of past iterations. This paper is intended to guide other instructors on how they might structure their programming foundations using the objects-first approach and our tooling.

In Abschnitt 2, we describe the size of our course and how it is organized. Abschnitt 3 explains briefly our used MPW. In Abschnitt 4, we discuss which concepts we teach and their sequence. Finally, we discuss our experience in Abschnitt 5, related work in Abschnitt 6, and conclude in Abschnitt 7.

2 Course Structure

Our course consists of two weekly lectures of 90 minutes, one weekly tutorial of 90 minutes (4+2 model), and one lecture hall exercise of 90 minutes. The course is worth 9 ECTS points, which amounts to a total effort of 270 hours for the students. The lecture takes place every winter term, and more than 800 students from more than 20 study programs are enrolled. Usually, the students are studying in their first semester, hence, without prior experience in programming besides their personal background. At the end, the students can evaluate the course (c.f., Abschnitt 5) and have to pass a written exam. This report is based on experience of four iterations of this course. In previous work, we have discussed a detailed description of the structural set-up during the Covid-19 pandemic [Sp22]. For a practical in-depth study of the content learned, students receive a weekly exercise sheet, which consists of an attendance part and a homework part. The students can work on the attendance part in groups of 2 or 3 onsite. Programming tasks are to be solved and discussed using pair programming. In addition, students can discuss and seek help from the tutor during the exercise. The homework part is to be solved independently in the same groups of two or three students and thematically relates to the topics of the previous week's attendance assignments. Each exercise sheet usually includes both programming tasks and theory tasks where students are expected to discuss topics with each other and the tutor to practice correct terminology. Furthermore, for student teachers, most exercise sheets contain additional exercises to discuss didactical questions [BBF20]. In addition, in the lecture hall exercise, the contents of the lecture are repeated, and questions from the students are discussed.

3 Objects-first with Mini Programming Worlds

Mini programming worlds provide a small, concise API that students can use without being directly overwhelmed. With the given API, students implement several small scenarios,

[3] `https://github.com/SQAHamster/plain-java-hamster`

Abb. 1: Overview of the topic sections in our lecture

which allow them to learn different concepts of OOP step by step. Especially in the objects-first approach, this offers the advantage of abstracting a lot of language-specific boilerplate code and complex concepts at the beginning and focusing on objects and their attributes as well as interaction between objects. However, many existing MPWs have limited support for a fully modern object-oriented API. For this reason, we have implemented a new object-oriented simulator in Java which is conceptionally based on the original hamster simulator by Boles et al. [BB04]. Objects in our simulator are, e.g., hamsters, grains, walls, and tiles on which the hamsters move and the walls lie. The hamsters can move one tile ahead, turn left, pick up, and put down grains, check if the tile in front of them is clear and check if there is a grain available on the current tile. Based on this, many scenarios of varying complexity can be programmed, such as solving a maze. Furthermore, OO concepts like inheritance can be integrated into the world, for example, to make the hamsters more powerful.

4 Content of the Lecture

In this section, we describe the content we cover in our lecture and its sequence and depth. In total, our lecture consists of 21 content lecture units, each of which goes over one or two lecture slots. The lecture units are structured in four topics sections as depicted in Abb. 1. In general, we discuss associated style guides for all concepts from Meyer's „Touch of Class“ [Me09] and other clean code rules.

Object & Class Foundations and Usage: (1) In the beginning, we discuss the motivation and introduction to computer science with the students. We then introduce elementary computer science concepts such as compilers and discuss the importance of software quality. (2) Then, we introduce students to the hamster simulator and form a basic understanding of programs. For this, we show simple examples of hamster games to demonstrate how programs can look like. We also decompose and format initial programs and call operations on a hamster object. In particular, we define objects and entities, discuss the difference between commands and queries and why a clear separation is important. We motivate object orientation for students by building stories through sequences of interactions between objects and their proximity to the real world. (3) In the second week, we discuss the structure of programs, addressing instructions and expressions, and defining lexis, syntax, semantics, and pragmatics of programs. In the process, students learn the steps of a compiler and build simplified abstract syntax trees for classes with operations only. In addition, we discuss static semantics rules. One objective of this lecture is to identify errors in program code and correctly classify them into error classes. (4) We then discuss the different types

of interfaces and their documentation. In particular, we focus on high-quality JavaDoc comments. Furthermore, we teach the concept of classes, as well as how to use classes and explain namespaces using packages as an example. A relevant part of this lecture unit is contracts, preconditions and postconditions of operations, and class invariants. (5) The fifth lecture unit deals with foundational logic.

Data Types, Variables, and Control Flow: (6) After learning the basics of using objects, we look at object creation, addressing object references and `null` and `this` references and their semantics. In addition, we look at the concepts of allocating, initializing, and assigning, explain constructors, and discuss the Java Optionals API. (7) To program more complex scenarios, we then teach control flow structures. In doing so, we begin with the concept of an algorithm and start with sequences. Then, students learn branches, the correctness of branches, and nested branches. For loops, we first teach the three normal types of loops without the foreach loop, dive into the correctness of loops via loop variants and invariants, and finally cover error handling. (8) In the eighth lecture unit, we discuss types and variables, as well as visibility scopes, releasing variables, and read-only variables. We also delve into primitive data types and discuss literals and type conversion. Furthermore, we discuss non-primitive data types and the difference between object equality (equals) and reference equality (same). Finally, we discuss how to make objects immutable.

Own Java Classes and Inheritance: (9) Lecture unit nine deals with programming own Java classes. We discuss static and non-static elements, functional decomposition, the uniform access principle, and access control via visibilities. Furthermore, we look at offensive and defensive programming and when to choose which variant. (10) After students know all the basic programming concepts, we discuss object-oriented inheritance and polymorphism. In doing so, we begin with the basics of inheritance and discuss the module viewpoint and type viewpoint. Afterwards, we focus on abstraction and interfaces, as well as overloading and overriding operations. Additionally, students will learn about polymorphism, static and dynamic types, and dynamic binding. Furthermore, type conformance, constructor chaining, access to operations and attributes of the parent class, and multiple inheritance are discussed. Finally, to ensure the correctness of inheritance, we explain Liskov's substitution principle to the students. (11) Following, we discuss concepts and definitions of various data structures, classify them into a taxonomy, and teach the foreach loop and iterators. We also go into more detail about the proper use of special collections and the map, and explain why arrays are usually to be avoided. (12) At this point, $2/3$ of the lecture period is over, and the Christmas lecture roughly outlines various programming languages.

Software Engineering Basics: (13) We begin the last third with a detailed discussion of clean code. (14) In addition, we discuss recursion with examples and go into recursive data structures, the correctness of recursion, and recursion elimination. (15) In the fifteenth lecture unit, we deal with modelling. Thereby, we introduce UML and particularly consider object diagrams, class diagrams, the relationship between class diagrams and implementation, as well as sequence diagrams. (16) Subsequently, we give a brief insight into various software engineering topics, such as development processes, requirements engineering, and software

architecture. (17) Further, we elaborate on the motivation and fundamentals of testing. Thereby, we discuss the identification of test cases, black box and white box testing, as well as automated (unit) testing. (18 & 19) Towards the end of the semester, we look at passing functions in OO languages, lambdas, inner and anonymous classes, method references, and the Java Streams API. (20 & 21) Finally, we look at program proofs, weakest precondition, and GUIs with events and listeners.

5 Discussion

Of the approximately 800 students who start in the lecture, roughly 550 students take the exam, with approximately 55% failing the exam, which seems a common rate for programming introductory courses in Germany. The exam is written on paper and consists of programming and concept exercises.

Furthermore, we evaluate each iteration of our lectures with a questionnaire that can be voluntarily completed online by the students. Due to the Covid-19 pandemic over the last few years, only approximately 100 to 150 students participated in the survey each time. In the questionnaire, students can rate various questions on a Likert scale from 1 (strongly agree) to 5 (strongly disagree) and provide free text responses for further feedback. There are separate surveys for the lecture, exercises, and lecture exercises. The questions primarily focus on the quality of organization, clarity of content structure, communication of learning objectives, comprehensibility of explanations, motivation, and addressing the students' concerns. The students rate the lecture between 2 and 2.3, the exercises between 1.8 and 2, and the lecture exercise between 1.3 and 1.8. Overall, therefore, students appear to be very satisfied with the course.

In the free-text answers, the students are particularly positive about the comprehensible and clearly well-structured content and the high amount of practical programming tasks with the hamster simulator. However, the students criticize the overall amount of material in the lecture, as well as the effort of the exercise sheets for students without prior knowledge. Likewise, students from technical engineering courses, such as mechanical engineering, would prefer to learn languages such as C++ instead of Java.

Furthermore, it is notable that students without previous knowledge in programming take up the objects-first approach very positively, while students, who had already learned to program with a bottom-up beginning, evaluate the objects-first beginning as negative. One problem with the approach is that students with previous knowledge get bored at the beginning and then stay away from the lecture but do not catch the right moment to join again. In addition, in some topics where students already have prior knowledge, we teach more complex content that students are unfamiliar with and thus miss, e.g., loop invariants or documentation of contracts. Nevertheless, we could not find a clear indication in the exam that having no prior knowledge leads to better results or vice versa. The use of the hamster simulator brings great advantages for students without prior knowledge due to its simplicity

at the beginning. Still, students often wish to use other MPWs later in the semester that allow more complex scenarios. Here, e.g., one could build an MPW that, instead of just having hamsters that can directly pick up grains, could work with raccoons whose worlds are more complex and must meet preconditions to open garbage cans.

In summary, an objects-first approach with MPWs is particularly useful for teaching object-oriented programming to students without prior experience. However, more complex scenarios and MPWs should ideally be enabled as the semester progresses, and multiple programming languages should be supported in the case of diverse courses. As a result, in addition to our hamster simulator, written directly in Java, we have built MPWs using MDSD frameworks [Fu21, FB21]. Currently, these can generate a hamster simulator in C++ and Java.

6 Related Work

Teachers often use the education IDEs BlueJ [Kö03, BKG06] or Greenfoot [Kö10, HK04] in their Java-based OO programming courses, e.g., [SZ07, GM08]. Both IDEs allow students to instantiate objects via a class diagram and interact with them graphically, thus offering a native objects-first approach. Additionally, both editors contain a source code editor with basic editing options but good visualization of block scopes. In our first three iterations, we used BlueJ in our first few exercise sheets before switching to industry-ready IDEs. However, BlueJ and Greenfoot currently do not run with newer Java versions (later than 11), and the students were fighting bugs in BlueJ on a regular basis. Therefore, we decided to add object manipulation directly to our hamster simulator and directly start with industrial IDEs. One reason to use IDEs such as Eclipse, IntelliJ, and Visual Studio Code as we want to teach our students more elaborate IDE features, such as refactorings, to prepare them for later courses and industry. Furthermore, we decided against them as we plan to support our hamster simulator in languages other than Java.

Schmolitzky et al. [SZ07] presented the structure of their programming introductory courses at the University of Hamburg. In general, they follow a similar approach. However, their course is divided into two semesters and has a different order of topics. For example, they include the implementation of data structures and algorithms in the first semester, which we exclude as there is another course at our university. Furthermore, our course includes basic SE concepts instead of focusing on programming only. Cooper et al. [CDP03] outlined their approach to teach objects-first with their own 3D MPW platform called Alice. However, their approach only teaches a scratch-like programming language, and they do not state the topics they cover in their course.

7 Conclusions & Future Work

In this paper, we described the structure of our programming introductory lecture and our experiences with the objects-first approach. In particular, we addressed the order and depth of various topics covered. Finally, taking into account student feedback from course evaluations over the past few years, we discussed when and how to teach programming using the objects-first approach. This paper intends to help other lecturers decide whether teaching according to objects-first is appropriate and what content can be taught within the available time. A major criticism of our implementation of the concept is that some students, especially those with previous experience, find our hamster MPW too boring after some time. For this reason, in the future, we plan to generate different worlds of varying complexity and programming languages through model-driven MPWs [FB21], thereby ensuring a more significant increase in complexity and variety.

Literaturverzeichnis

[BB04] Boles, Dietrich; Boles, Cornelia: Objektorientierte Programmierung spielend gelernt mit dem Java-Hamster-Modell, Jgg. 1. Springer, 2004.

[BBF20] Becker, Steffen; Bescherer, Christine; Fest, Andreas: Reflective Pedagogical Practice on and in Introduction to Programming and Software Engineering. In: Tagungsband des 17. Workshops SSoftware Engineering im Unterricht der Hochschulen"2020. Jgg. 2531 in CEUR Workshop Proceedings. CEUR-WS.org, S. 7–10, 2020.

[BKG06] Barnes, David John; Kölling, Michael; Gosling, James: Objects First with Java: A practical introduction using BlueJ. Pearson/Prentice Hall, 2006.

[CDP03] Cooper, Stephen; Dann, Wanda; Pausch, Randy: Teaching objects-first in introductory computer science. Acm Sigcse Bulletin, 35(1):191–195, 2003.

[FB21] Fuksa, Mario; Becker, Steffen: Mini Programming Worlds: Teaching MDSD via the Hamster Simulator. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C). IEEE, S. 696–701, 2021.

[Fu21] Fuksa, Mario: Redesigning the Hamster Simulation. Masterarbeit, University of Stuttgart, 2021.

[GM08] Gallant, Randy J; Mahmoud, Qusay H: Using Greenfoot and a Moon Scenario to teach Java programming in CS1. In: Proceedings of the 46th Annual Southeast Regional Conference on XX. S. 118–121, 2008.

[HK04] Henriksen, Poul; Kölling, Michael: Greenfoot: combining object visualisation with interaction. In: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. S. 73–82, 2004.

[Kö03] Kölling, Michael; Quig, Bruce; Patterson, Andrew; Rosenberg, John: The BlueJ System and its Pedagogy. Computer Science Education, 13(4):249–268, 2003.

[Kö10] Kölling, Michael: The Greenfoot Programming Environment. ACM Transactions on Computing Education (TOCE), 10(4):1–21, 2010.

[Me09] Meyer, Bertrand: Touch of Class, Jgg. 51. Springer, 2009.

[Sp22] Speth, Sandro; Krieger, Niklas; Reißner, Georg; Becker, Steffen: Teaching during the Covid-19 Pandemic — Online Programming Education. In: Software Engineering im Unterricht der Hochschulen (SEUH 2022). Gesellschaft für Informatik, S. 89–94, Februar 2022.

[SZ07] Schmolitzky, Axel; Züllighoven, Heinz: Einführung in die Softwareentwicklung-Softwaretechnik trotz Objektorientierung? In: SEUH. S. 87–100, 2007.