Rony G. Flatscher (2009):
An Outline for a Syllabus for Introducing End-user Type of Students to the Object-oriented Paradigm.
In Christopher Scaffidi, Gunnar Stevens (Eds.),
International Reports on Socio-Informatics (IRSI),
Proceedings of the Work-in-Progress Session of the Second International Symposium on End User Development
(Vol. 6, Iss. 1, pp. 13-17)

An Outline for a Syllabus for Introducing End-user Type of Students to the Object-oriented Paradigm

Rony G. Flatscher
Wirtschaftsuniversität Wien, Institut für Betriebswirtschaftslehre und Wirtschaftsinforma-tik, Augasse 2-6, A-1090 Wien, Austria
rony.flatscher@wu-wien.ac.at

Abstract. This work-in-progress paper sketches a syllabus for introducing end-user type of students at the Wirtschaftsuniversität Wien to the object-oriented paradigm. The knowledge of this syllabus then serves as the fundamental building block for subsequent syllabi for scripting Windows and Windows applications and for scripting Java and Java applications.

Keywords: EUD (End-user Development), EUP (End-user Programming), Ob-ject-oriented Paradigm, Syllabus, ooRexx.

Introduction

Working at a University (Wirtschaftsuniversität Wien, WU) where more than 20,000 students study Business Administration and Economics, there are quite a few students who are very interested in Business Informatics/Management Information Systems (MIS) but have either no or poor prior exposure to programming. As today's software infrastructure is heavily based on object-oriented (OO) concepts, it is mandatory to teach students the OO basic concepts and have them apply their acquired knowledge on a regular basis, such that many of the abstract concepts become "tangible" for the solution of (e.g. Business process) problems with the help of a programming language or environment.

This work-in-progress paper introduces the syllabus for teaching WU-students the basics of programming and in the process concentrates on the OO-paradigm. The pro-gramming language used in those classes is “Open Object Rexx” (ooRexx, cf. [1, 2, 3, 4]), which can be regarded as a “human centric”, basically typeless, interpreted pro-gramming language, which is available for practically all operating systems. It im-plements all of the most important OO concepts and therefore can be used to demon-strate and experiment with these.

1 Set-up of the Lecture and the Syllabus

This syllabus accounts for two European Credit Transfer System (ECTS) points. The students are set up into groups of two, such that no one is left on his/her own, when creating the assigned (homework) programs. After each installment the students must create two small programs on their own, which each stress some newly introduced concept. The homework has to be turned in via a mail server list such that all other students are able to see and study the homework of their colleagues one day before the next installment of the class takes place, i.e. within seven days. That mail server list is also intended for seeking and giving help among the students (although rarely used for that purpose, probably because students do not want to document in the public that they would not understand some fundamental concept).

As one cannot learn how to swim in a classroom only, it is mandatory that the students “wet their feet” and finally start to learn swimming in the water, it is important for end-users to really apply the theoretically learned concepts with a real programming language. Such a programming language should be easy to learn (i.e. possesses among other things a simple syntax) and easy to debug (i.e. give as helpful error messages as possible and allow for debugging at various levels of detail). Over the course of many years the author “stumbled” over a practically unknown scripting language that was originally created by IBM, Object REXX, that nicely realizes the aforementioned important properties. (That language was donated by IBM to the RexxLA and is now a free and open source scripting language, named “Open Object Rexx (ooRexx)”.) Experimenting with this OO-scripting language at the University of Es-sen in the beginning of this millennium, and then later at the University of Augsburg and finally at the Wirtschaftsuniversität Wien, yielded a very helpful teaching tool that demonstrates the taught (OO) concepts very nicely. All interested students, Business Informatic students in the case of Essen, and Business Administration students (“end-user-developers/programmers”) alike could master the language, and more important the OO concepts within five weeks à four lecture hours, which has been very remarkable and helpful for the subsequent courses that build on the results of this one.

1.1 Syllabus for the Foundation of Programming

The first part of this course introduces the building blocks of programming, like the definition of a statement, flow-of-control (repetitions, selections, procedures/functions, modules/packages), and the runtime environment, under which pro-grams get executed.

1.1.1 Installment 1

An overview of the course is given, followed by the thoughts that led to picking the ooRexx language as the tool for this course. A brief history of ooRexx is given, pointing out the motivation of creating it in the first place and discussing design goals like "human-centricness" of the programming language, which makes it so suitable for end-user-development/programming.

Confronting the students with a short hello-world program in ooRexx it is explained, that the program is stored as a plain text file and needs to be interpreted by the ooRexx interpreter. In this context the online help system is introduced and explained, stressing the excellent reference PDF documentation that comes with the language.

The students will learn the concepts of a variable, a statement, a block, a branch, and repetition (loop). As ooRexx is not strictly typed there is no need to explain types at this time at all.

1.1.2 Installment 2

The students learn about labels which are used as jump targets and that are needed, if one organizes repetitive code as procedures and functions. In addition built-in functions (BIF) are introduced as well as external programs serving as jump targets for procedures and functions. For this to work reliably, resolution rules need to be de-fined, that are followed by the interpreter.

Creating more complex programs by creating procedures and functions is eased by the concept of a scope, which allows for insulating the variables of a procedure or function from the rest of the program.

This installment continues with the introduction of the concept of associative arrays, dubbed "stem"-variables in ooRexx, and concludes with the keyword instruction "PARSE" which makes it very easy to parse strings into different parts.

1.1.3 Installment 3

In this installment the students learn about the concept "conditions", "exceptions" and how to intercept them, if the programmer so desires. In addition retrieving arguments by reference within procedures and functions is discussed, making it for the first time explicit that so far only calls by values got carried out.

The concept of "directives" is introduced, which are carried out by the interpreter prior to executing the program. A brief overview of the directives "requires", "routine", "class" and "method" is given, followed by a more thorough discussion of the "requires" and "routine" directives.

1.2 Syllabus for Object-oriented Programming

The second, concluding part of this course introduces the OO concepts of class, sub-classing/specializing, class hierarchy, inheritance including multiple inheritance (!), methods, method resolution (and the role of the variables named "self" and "super"), message (available as first class objects, FCO, in ooRexx, as well as the "un-known/cannot-understand-message"-concept) and some of the most important, common collection classes for the utilities, they offer the programmer.

1.2.1 Installment 4

First the concept of an "abstract data type (ADT)" is introduced and the concepts "at-tribute" and "function/behaviour" are introduced and discussed. Object-oriented programming languages are designed to easily implement ADTs in the form of classes. It is stressed that the OO-paradigm introduces an own set of "termini technici" like "class", "object/instance/entity", "inheritance", and he like.

With the help of simple examples the correspondence between ADTs and their implementations in the form of classes – in ooRexx using the "class" and "method" directives – is repeatedly given. Taking advantage of classes necessitates the knowledge of the concepts "messages", "cascading messages" and the introduction of additional scoping rules.

The concepts of "constructor" and "destructor" get explained and their effects demonstrated with little nutshell examples.

This installment concludes with the introduction of the concept of a "classification tree" and how it gets used in method resolution, introducing the runtime variables "self" and "super" in this context.

1.2.2 Installment 5

Firstly, the OO-concepts introduced in the prior installment get repeated, followed by a detailed explanation of the "class" and "method" directives which allow for re-iterating the important building blocks.

Following the class hierarchy explanations the concept of "multiple inheritance" is introduced and exemplified with a little nutshell example. (This originally was moti-vated by reading an interview where the creators of Java thought that this concept is difficult/not understood by developers and therefore error-prone. Every EUD so far was able to understand that concepts without any problems!)

This installment concludes by introducing and characterizing the core class hierarchy that comes with ooRexx, concentrating on the collection classes.

2 Conclusion and Outlook

This article briefly introduced a syllabus for teaching end-user-kind of students (i.e. Business Administration and Economic students) the object-oriented concepts and object-oriented programming with the help of an easy to learn scripting language named ooRexx in a course of two ECTS points.

Building on this fundamental building block it becomes then possible to teach such EUD-kind of students in another two ECTS points class the fundamentals of scripting Windows and Windows applications (even beyond Microsoft Office!) using the infra-structural ActiveX interfaces and Windows script host (WSH). Explaining the ooRexx OLE proxy class for OLE-driven scripts is then a matter of 20 minutes only!

In addition it becomes possible for EUD with the OO-building block knowledge in four ECTS point course to teach scripting of Java and Java applications, which allows for creating operating system and platform independent scripts and applications!

It would not be possible to achieve the same depth of a working knowledge for EUD with a programming language like Java, C++, C#, Perl, Python, or even Visual Basic (or VB.Net for that matter), because of the syntax rules, richness of functionalities and (syntax) peculiarities of those languages.

3 References

1. Cowlishaw, M.F.: The REXX Language. Prentice Hall, Englewood Cliffs (1990)
2. Flatscher, Rony G.: 2006. Resurrecting REXX, Introducing Object Rexx. RDL Workshop, ECOOP 2006, Nantes, Frankreich, 3.7.-7.7. (2006). URL (as of 2009-01-19): http://prog.vub.ac.be/~wdmeuter/RDL06/Flatscher.pdf
3. Flatscher, Rony G.: Slides “ooRexx_1.pdf” through “ooRexx_6.pdf” introducing the OOo paradigm at the Wirtschaftsuniversität Wien (WU). URL (as of 2009-01-19): http://wi.wu-wien.ac.at/rgf/wu/lehre/autowin/material/foils/
4. Fosdick, H.: Rexx Programmer’s Reference.Wiley Publishing, Indianapolis (2005)
5. Homepage of Open-object Rexx (ooRexx), http://www.ooRexx.org