

# **Proceedings of the Seventh Annual Java & the Internet in the Computing Curriculum Conference**

## **JICC 7 Monday 27th January 2003 London Metropolitan University**

9.30-10am Registration & Coffee **including** 9.55-10am Opening

10-10.30am **A simple Interactive Development Environment for C#**

Trevor Fenner, George Loizou, Keith Mannock and Marie-Helene Ng Cheong Vee, School of Computer Science and Information Systems, Birkbeck, University of London

10.30-11am **A Java IDE published under the GNU licence**

David Mole, School of Computing, Information Systems and Mathematics, South Bank University

11-11.30am Coffee and demonstrations **including** Text Books by Leading Publishers

**Java Learning Objects** - Prof. Tom Boyle, London Metropolitan University

**Free Text Plagiarism Tool** - Fintan Culwin, South Bank University

11.30-12pm **Teaching Java, XML and e-Commerce: Some reflections of past and future approaches from the “front-line” at Luton University 1999-2002**

Tim French & Des Stephens, Dept. of Computing and Information Systems, University of Luton

12-12.30pm **Web Services and Java**

Chrisina Draganova, Department of Computing, Communication Technology and Mathematics, London Metropolitan University

12.30-1pm **Designing a reusable Java Teaching module**

Safia Barikzai, School of Information Technology and Computer Science, University of Wollongong

1-2pm Lunch and demonstrations (as above)

2-2.30pm **The Java Infrastructure for Rart, a New Art Form**

Jan Aminoff, RART

2.30-3pm **Teaching Java programming to media students with a liberal arts background**

Jens Bennedsen, Department of Computer Science, University of Aarhus

3-3.30pm Coffee and demonstrations (as above)

3.30-4pm **eXtreme Programming (XP) with Java and Jython in the Classroom**

David Dench, Huddersfield University

4-4.30pm **A Distance Learning OOPs Course Based on Java**

T. F. Higginbotham, Southeastern Louisiana University

4.30-4.45pm Commercial Presentations (IBM, Microsoft & Sun invited).

Chair: Peter Chalk,

Department of Computing, Communication Technology and Mathematics,  
London Metropolitan University.

Organised by the LTSN-ICS, University of Ulster, in co-operation with the ACM SIGPLAN & SIGCSE.

## JICC 7 Introduction by Peter Chalk, Conference Chair

Welcome to the seventh Java & the Internet in the Computing Curriculum Conference, previously just about Java, but now including additional topics of interest to the community.

We start with a paper by **Trevor Fenner** *et al* on a C# IDE. Last year, Barry Cornelius led us through C# as a contender for an introductory programming course, so it seems appropriate to start discussing a suitable development environment. A similar motivation underlies the paper by **David Mole**, who is presenting a Java IDE intended for student use, under the GNU public licence scheme.

**Tim French & Des Stephens** cover the teaching of XML to students studying an eCommerce style course. Web Services, based on XML, is also the topic addressed by **Chrisina Draganova**, who presents syllabi suitable for both business and computing courses.

Reusable learning objects is an emerging area of interest across all subject domains and JICC 7 includes both a paper by **Safia Barikzai** and a demonstration by **Tom Boyle**, on Java learning resources. Another demonstration during the breaks will be given by **Fintan Culwin** (who organised the first 5 JICC conferences at South Bank University) on a free text plagiarism tool.

Introducing programming using a graphics library has always been popular and **Jan Aminoff** presents an API for computer-generated art, which might open up the subject to wider audience. Such an audience, media/liberal arts students, is considered by **Jens Bennedsen**, who evaluates what such students actually want from a Java course.

Again the problem of teaching introductory programming is addressed by **David Dench**. He suggests the use of Extreme Programming (XP) to raise motivation. Delivering a course to distance learning students can also be problematic - **Tom Higginbotham**, a regular JICC contributor from Louisiana, explains how he solved this and what issues are raised, such as the additional work load for staff.

Finally, we hope to see the usual 'debate' between representatives of the big three players in the world of Java and the Internet - Sun, IBM and Microsoft - who may supply the odd freebie, if we are lucky.

# A Simple Interactive Development Environment for C#

*Trevor Fenner, George Loizou, Keith Mannock and Marie-Helene Ng Cheong Vee*

School of Computer Science and Information Systems,  
Birkbeck, University of London  
London, WC1E 7HX



## Abstract

We discuss the C#ide ("seaside") system, a lightweight, easy-to-use teaching environment for the C# language that facilitates the teaching of C# on introductory programming courses. As with similar systems (e.g., BlueJ), we have placed emphasis on the visualisation and interaction techniques. What differentiates C#ide from other systems is the additional functionality of the system, which provides a highly interactive environment that encourages experimentation and exploration whilst not significantly reducing functionality. The objective is to develop a system which has "more mileage" than systems like BlueJ but significantly reduces the learning curve and transitional problems associated with migrating to more complex integrated development environments (e.g., Visual Studio .NET).

## 1. INTRODUCTION

We discuss the motivation for C#ide, a simple interactive development environment (IDE) for C# and .NET, and a prototype design and implementation. It is well known that teachers of object-oriented programming courses at the introductory level face numerous problems in course design and presentation. C#ide aims to provide an environment for C# and .NET that is similar in approach to that provided by BlueJ<sup>1</sup> for Java, and yet has the functionality of systems such as #develop<sup>2</sup>. The Visual Studio .NET environment is a comprehensive tool-set for rapidly building and integrating XML Web services, Microsoft Windows-based applications and Web solutions. For students on introductory programming courses, this environment appears to be large and complex.

In the following sections we will describe the motivation for the development of C#ide, the initial design, the state of the implementation to date, and our initial impressions of the system. Where appropriate we compare and contrast C#ide with BlueJ and #develop, the systems that are closest to ours in terms of approach and functionality. (We have not included eclipse<sup>3</sup> in this comparison as it doesn't have the graphical interface of BlueJ and, at the time of writing, doesn't have a C# mode.)

## 2. MOTIVATION

Initially we posed ourselves the questions:

- why develop yet another IDE when good systems already exist (e.g., Visual Studio .NET and #develop)?
- why should we consider teaching C# when our Java courses are well established and we have considerable experience and knowledge in the area?

### 2.1 Why Another IDE?

Much in the same way as with BlueJ, we saw the need for a lightweight, easy-to-use teaching environment for the C# language. The Visual Studio .NET IDE has been developed over many years and is the de facto standard for application development on the Microsoft Windows platform. The main problem from a teaching viewpoint (now that licensing ceases to be such an issue) is that it is complex and daunting for students. Once the initial complexity is learnt then the functionality enables the student to be much more productive (and adds another “skill” to the CV), but as with most tools, it often masks the underlying concepts.

So why not just produce a “clone” of the BlueJ environment for another language, i.e., C#? After all, there has been widespread acceptance of BlueJ<sup>4</sup>.

- Over many years of teaching object-oriented programming, we have become aware of the visual metaphors that students find most helpful for learning object-oriented concepts. For this reason we believe the environment should display the class structure visually<sup>5</sup>.
- When teaching object-oriented design and programming, we spend most of the time discussing class structure. Students typically experience difficulty thinking in terms of classes and objects (especially if they have previous experience in using procedural-based programming languages); yet structuring the problem via these concepts is one of the most important aspects of object-oriented programming.
- The visualisation process is made much easier if students do not just see lines of code but also have a design view of the software they are developing (cf. Rational Rose<sup>6</sup>).

These are all desirable aspects of an IDE system, but some of the drawbacks we have noted with BlueJ-like systems are:

- that the knowledge of using a specific IDE with reduced functionality does not necessarily transfer to a more feature-rich environment. Our students start off with BlueJ for their Java programming, but have to migrate to NetBeans (Sun ONE Studio)<sup>7</sup> or a similar IDE for more complex programming work (e.g., J2EE, GUI work, etc). This requires explicit instruction to support the move and it is often necessary to supplement this with appropriate exercises;
- that it may be reasonably easy for students to tackle exercises in which they have to modify existing source code (facilitated by the programming environment’s code templates and stubs), but it is not an easy transition for them to write classes from scratch.

This can lead to wasted time and sometimes complete confusion for the student. This also occurs if they have to work in a command shell environment; what they have learnt does not easily transfer. Simple things like constructing an object often cause the students many syntactic and semantic problems.

## 2.2 Why C#?

There are two points to consider when we talk about adopting a new programming language. One is based on pedagogy: does the language have feature *XX*, etc.? C# does have several features that are an improvement upon/different from Java<sup>8</sup>. The other is a marketing viewpoint: it is impossible to ignore the dominance of Microsoft related products in the current computing environment. The majority of our students study part-time and work in the “real world”. They see Microsoft products every day and are accustomed to using them. In an ideal world we would probably be using Smalltalk, Eiffel, or a similar language, to teach object-oriented programming. We do not operate in such a rarified atmosphere and therefore we have to make our courses appealing to prospective students; both C# and .NET have received considerable exposure in the media and it would be unreasonable for us not to consider them.

Our programming language teaching currently uses a combination of C++ and Java. The current dual language approach often leads to confusion due to the existence of similar language concepts that may, however, have different syntax or, more importantly, different semantics. The usual problems raise their head with C++: pointers, memory allocation, programming in the large, etc. Of course Java does not solve all the problems, but it is easier to make the transition from C# to Java than from C++ to Java – assuming that only managed code is allowed<sup>†</sup>. If similar teaching environments are used then the “compare and contrast” approach to the languages becomes much simpler, and hopefully the students’ comprehension of the language features is improved<sup>9</sup> – this remains to be demonstrated however.

## 3. DESIGN

One of the fundamental requirements is that we do not want to have to devote a significant amount of time to teaching about the environment itself. Special emphasis has been placed on visualisation and interaction techniques to create a highly interactive environment that encourages experimentation and exploration. As we have already stated, this environment is similar in ethos to the BlueJ<sup>10</sup> tool for the Java programming language. The tool has proved popular with our existing students and a tool with similar functionality would ease the migration to C# and .NET.

One of the fundamental concerns when designing the environment was that it be object-oriented. In C#ide we are building an environment in which classes and objects are the basic components of interaction. In this way, the students are led straight into the object-oriented paradigm in terms of classes and objects. C#ide is highly interactive and allows students to easily experiment with their code/design. The ability to create objects “on-the-fly” and operate directly upon them is a valuable aspect of the learning process. We have found that if students can create an object by a couple of mouse clicks, interrogate its structure directly, and call methods, they grasp the essentials of object-oriented programming far more quickly.

Figures 1 and 2 illustrate the different approaches in interface provided by BlueJ and #develop, respectively; #develop’s increased functionality is reflected in the complexity and “clutter” of the display. This contrasts with the simplicity and more user-friendly appearance of BlueJ.

---

<sup>†</sup> Non-managed code does, however, permit the possibility of the low level (C++ like) programming that is required for certain applications, e.g., advanced memory management and garbage collection.

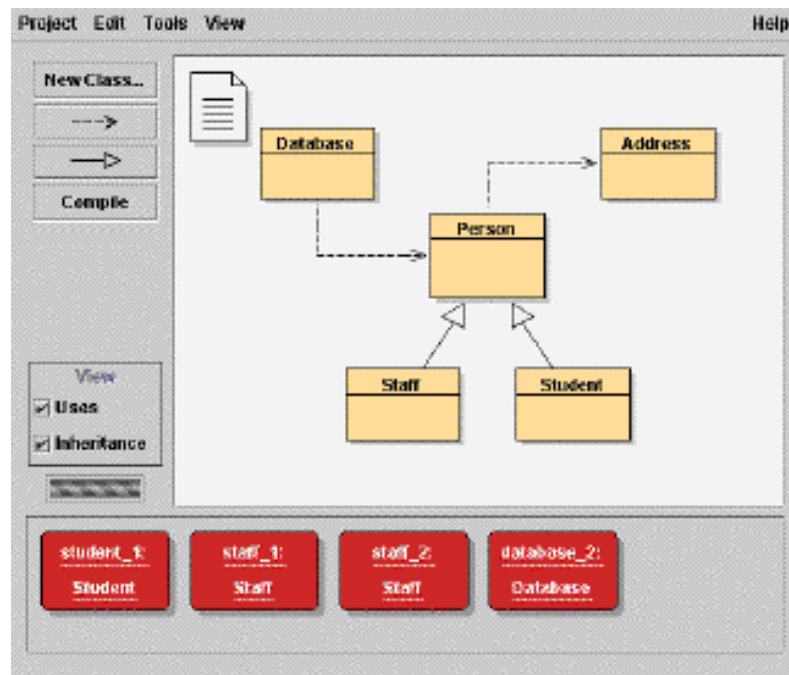


Figure 1: The BlueJ workbench

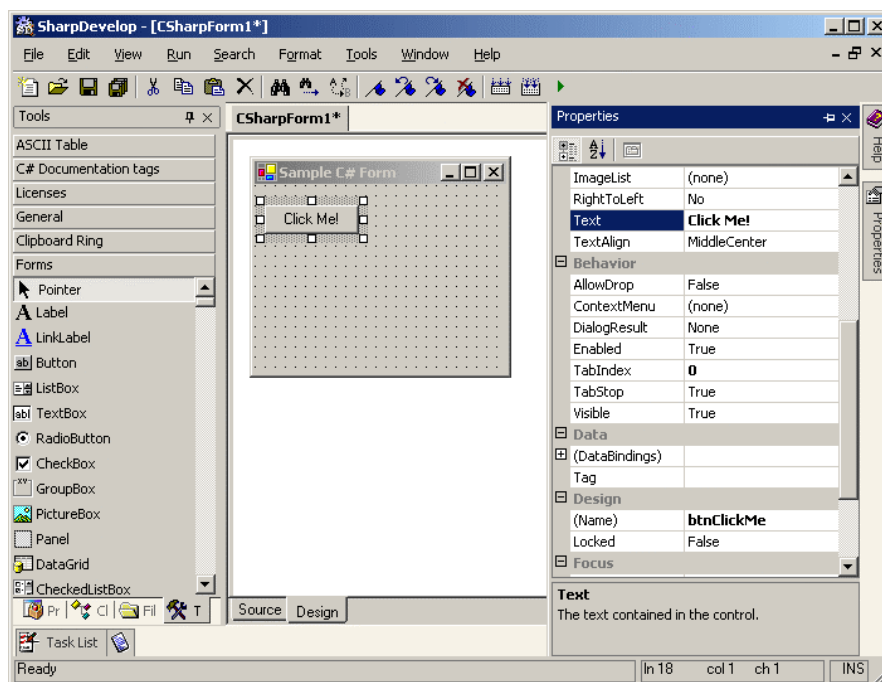


Figure 2: The #develop workbench

The prototype system is a fully integrated environment with the following characteristics:

- a simple project manager
- a language sensitive editor
- graphical display of the class structure
- compilation facilities for C#
- access to the underlying Common Language Infrastructure (CLI)
- an *objectbench* with creation, interrogation and method invocation facilities

- a two-tier debugger that provides a simple single step tracing capability and inspection facilities, together with a more advanced mode
- “hooks” to enable the developer to extend the functionality of the system
- access to source control software (for version management)
- a simple to use, yet comprehensive, user interface targeted at beginners
- easy access to language reference and help facilities.

#### 4. IMPLEMENTATION

`C#ide` is written in C# for ease of development, efficiency, and as a demonstration of the functionality and flexibility of the C# language and .NET environment. If this system were solely available in a Microsoft environment then its usefulness would be severely limited. We have students that use a variety of platforms, so support for multiple operating systems is essential. Platform independence should also aid the adoption of C#, which otherwise is based solely in the Microsoft world.

To facilitate this process we have used the Rotor package<sup>11</sup>, which provides a free, shared-source implementation of Microsoft’s Common Language Runtime (CLR). The package includes source code for the C# compiler, as well as for the CLI itself. In particular, it contains source code for a variety of useful developer tools, including a Common Intermediate Language (CIL) assembler (`ilasm`), a disassembler (`ildasm`), a debugger (`cordbg`), a profiler and an assembly linker, all of which we intend to incorporate into `C#ide`. The current release of the build runs on Windows XP, the FreeBSD operating system, and Mac OS X 10.2. Whilst this doesn’t give us total coverage of the possible platforms, it has enabled us to provide hooks into tools such as `cvs`, `nmake`, `build`, and `ant`. (These are not required for students at the introductory stage, but should increase the longevity of `C#ide`’s usefulness.) It also provides a good *proof of concept*.

The major problem with using Rotor as the basis for our implementation is that it does not include the Windows Forms classes. This has meant that the graphical user interface has initially been limited to the Windows XP environment until the design and implementation of an appropriate cross-platform graphical library is finished. There have been several suggestions as to how this might be achieved, some utilising Tk/Tcl<sup>12,13</sup> others using the SWT (Standard Widget Toolkit)<sup>†</sup> or XUL (XML-based User Interface Language)<sup>14 ‡</sup>. Both XUL ([Netscape and Mozilla Public Licenses](#)) and SWT ([Common Public License](#)) are open source cross-platform technologies, so they can be used freely in `C#ide`. SWT is currently available only for use with Java, so additional work would be required to create .NET/CLR compatible versions - but this would still be considerably easier than trying to do the same for Windows Forms.

At the time of writing we have implemented a prototype system which provides:

- the compilation of C# in the IDE without any additional setup required from the user (i.e., assemblies, projects, etc.)
- a text editor with code completion facilities and syntax colour-coding functionality
- a lightweight IDE linking into many of the Rotor facilities (we have avoided solely Microsoft Windows-based features wherever possible)
- visualisation of the classes

---

<sup>†</sup> This was created by IBM and integrated with the Eclipse development environment.

<sup>‡</sup> This uses an XML-formatted language to define a *description* of the interface objects.

- an experimental object workbench enabling creation, interrogation and method invocation facilities
- integration with the underlying debugger.

## 5. INITIAL IMPRESSIONS

We are about to test a prototype version of `C#ide` with a group of students who are taking the Object Oriented Programming module on our “conversion” MSc course. They have a mixture of Java and C# tuition on this module, utilising `BlueJ` for the first part of the course. A report on the findings will be published in due course.

Our initial impressions are that the prototype `C#ide` has most of the functionality and ease of use of `BlueJ` coupled with some of the additional features of `#develop`. The current version of the prototype only works under Windows XP using Rotor, as the portable graphical classes are currently under development. We have achieved most of the functionality outlined in Section 3, but we intend eventually to

- add the ability to write not just C# code but, if possible, to incorporate ASP.NET, ADO.NET, XML and HTML into the environment
- improve project development and add source code control system support
- improve the syntax colour-coding for C#
- add the ability to place bookmarks in the code
- add support for limited code templates (too many wizard options only confuse the students and mask what is actually going on)
- incorporate regular expression matching into the find and replace functions
- increase the documentation generation facilities
- attempt to integrate `C#ide` with `#develop/eclipse`.

As the project progresses we intend to consider the use of `sourceforge.net` as a software repository as we would like `C#ide` to be open source (unlike `BlueJ`). We believe this approach would give confidence to users concerned about being tied to Microsoft environments. The current licensing of Rotor needs to be investigated to determine whether this would be possible<sup>15</sup>.

## ACKNOWLEDGEMENTS

This pilot project has been supported in part by a grant from *Microsoft Research*.

## REFERENCES

---

<sup>1</sup> Kölling, M., Teaching Object Orientation with the Blue Environment, *Journal of Object-Oriented Programming*, 12(2): 14-23, 1999.

<sup>2</sup> [<http://www.icsharpcode.net/OpenSource/SD/default.asp>]

<sup>3</sup> [<http://www.eclipse.org/org/index.html>]

<sup>4</sup> List of institutions using BlueJ.

[<http://www.bluej.org/users/users.html>]

<sup>5</sup> Hosking, J.G., Visualisation of Object-Oriented Program Execution, *Proceedings 1996 IEEE Symposium on Visual Languages*, 190-196, 1996.

---

<sup>6</sup> [<http://www.rational.com/>]

<sup>7</sup> The NetBeans™ Tools Platform: A Technical Overview. 2001.  
[<http://www.sun.com/software/sundev/whitepapers/netbeanstp.pdf>]

<sup>8</sup> Cornelius, B., Comparing .NET with Java, *Java & the Internet in the Computing Curriculum Conference (JICC 6)*, 2001.

<sup>9</sup> Albahari, B., A Comparative Overview of C#, 2000.  
[[http://genamics.com/developer/csharp\\_comparative.htm](http://genamics.com/developer/csharp_comparative.htm)]

<sup>10</sup> Barnes, D.J. and Kölling, M., *Objects First with Java: A Practical Introduction using BlueJ*, Prentice Hall / Pearson Education, 2003.

<sup>11</sup> The Shared Source Implementation of the CLI and C# (“Rotor”), 2002.  
[<http://research.microsoft.com/programs/europe/rotor/>]

<sup>12</sup> Bishop, J. and Horspool, N., An Independent GUI Development Tool on Rotor, *First Rotor Project Workshop*, Queens' College, Cambridge, 23-26 July 2002.

[<http://research.microsoft.com/programs/europe/rotor/wshop-viewstalk.pdf>]

<sup>13</sup> Jacques, F. and Batista, J-C., *Rotor: You Spin Me Right Round*, 2002.

[<http://www.macadamian.com/column/tinymessenger.html>]

<sup>14</sup> Jones, A.R., *SWT, XML Put True Cross-platform GUIs Within Reach*, 2002.

[<http://www.devx.com/xml/Article/9782/0/page/1>]

<sup>15</sup> Mundie, C. and Tiemann, M., *Shared Source vs. Open Source*, 2001.

[[http://linux.oreillynet.com/pub/a/linux/2001/08/09/oscon\\_debate.html](http://linux.oreillynet.com/pub/a/linux/2001/08/09/oscon_debate.html)]

# A Java IDE published under the GNU licence.

David Mole

School of Computing, Information Systems and Mathematics  
South Bank University

## 1 Introduction

The commercial IDEs available for teaching Java, like much computer software, tend to be over-endowed with unwanted features, which make them unnecessarily difficult for students to use. There are some freely downloadable IDEs, mostly copyrighted and without source code. These are a mixed bunch, a few work well but many fall over very quickly and most have limited feature-sets. So, if one prefers not to use a commercial IDE, then it is difficult to find a product that fully satisfies ones requirements. How nice it would be if there were a reliable basic IDE freely downloadable with its source code so that it could easily be extended to meet different requirements.

It therefore seems worthwhile to ask if the Java-teaching community would be interested in developing software under the GNU General Public Licence (1) that, like other open-source software, could gradually be improved by communal effort. For such a development to occur, the source code must be published. This paper describes the development of a Java IDE (JGIDE) that will be published in January 2003 under the GNU GPL

The requirement satisfied by JGIDE is described in section 2. Section 3 briefly describes its structure and implementation. Section 4 contains an abbreviated description of the facilities implemented by JGIDE with some screen shots and some additional remarks by way of justification for the facilities included in, or omitted from, JGIDE. Section 5 compares JGIDE with some other Java IDEs and Section 6 provides a short discussion and conclusion.

## 2 Requirement

Learning how to use Java is hard enough without burdening the student with a complex IDE. So the requirement was for a minimal Java IDE that provides basic facilities. The IDE was to be written in Java so that it could be run on various platforms with little or no modification. The implementation was to make use of the Java Class Library as far as possible.

The IDE was to enable the user to open, edit, save, print, compile and run Java programs. It was to be possible to have several Java source-code documents open concurrently. The results of compilation and execution of Java programs were to appear in windows separate from the source code (to facilitate error correction).

The editing requirements were fairly standard, including line-numbering, cut, copy, paste, undo, redo, find, replace, select-all, cursor movement with the arrow keys, selection of text by double-clicking, etc. To these minimal requirements I added a couple of my favourites, namely automatic completion of quotes and brackets (to reduce some common errors) and re-indentation of code (to make it easier for me to read students' work).

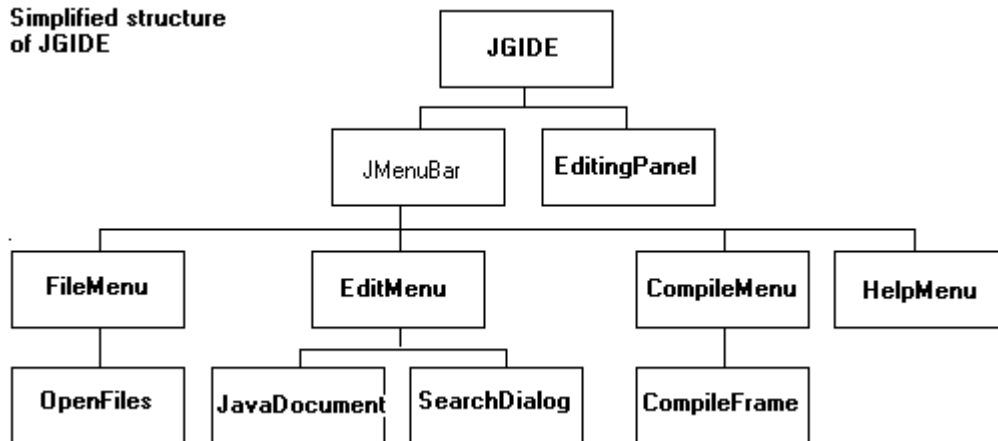
The requirement was to use the standard Sun compiler and interpreter.

## 3 Design and implementation

Figure 1 shows a slightly simplified version of JGIDE's structure. The diagram shows JGIDE as a system composed as a hierarchical structure of various components. The boxes show the name of the class from which each component is constructed. The simplification consists of the omission of some standard Java classes from the diagram: thus the EditingPanel is a component of a JScrollPane and EditingPanel itself is composed of two JTextAreas, one of

which is the editing area, the other is used for line numbering. This structure was chosen to reduce coupling between the parts of the system.

**Simplified structure of JGIDE**

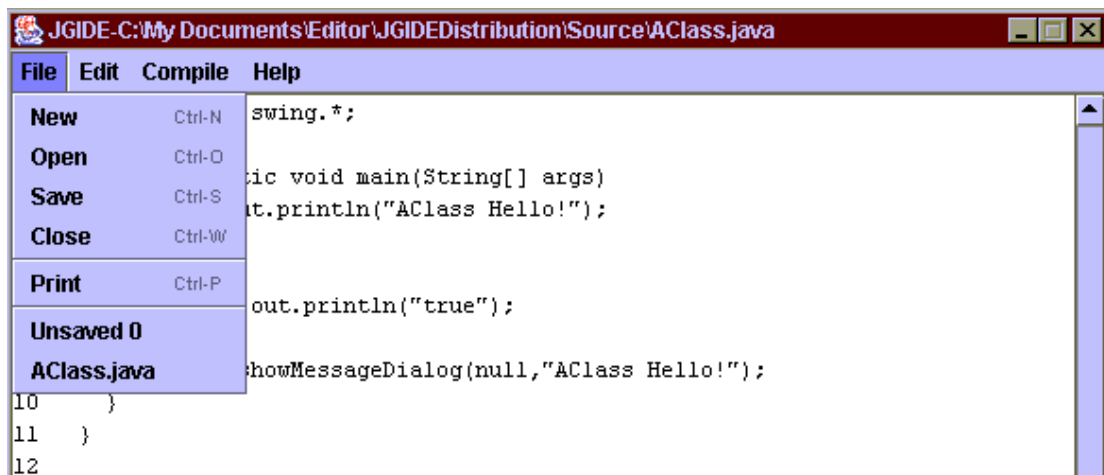


**Figure 1.**

At the time of writing, the implementation of JGIDE consists of 10 classes totalling less than 1100 lines. The detailed implementation of these classes is not discussed here: the source code will be available from my website in January 2003 (See section 7).

#### 4 Outline of the facilities currently implemented by JGIDE

JGIDE starts up with a new empty file open for editing. JGIDE provides a minimal set of facilities: it enables the user to open, edit, save, print, compile and run Java programs. Each facility can be invoked either by selection from a menu or by using a Ctrl-Key combination. The JGIDE application uses a stateless menu-driven user interface. There are four menus - File, Edit, Compile and Help, which are described next.



**Figure 2.**

The file-menu contains five permanent items (New, Open, Save, Close and Print) and a variable list of the files open in the IDE. A document may be selected for editing from the list by clicking on its name in the usual way. When the user selects a file it is displayed in the edit-area and its path-name appears at the top of the JGIDE window, see Figure 2. When a file is selected, the text of the previous file is preserved and can be restored for editing simply by clicking on its name in the list. The number of files that may be put on this list is limited only by the size of the screen.

The "Open" and "Save" options display file dialog boxes: the files displayed are all whose names end in ".java" or ".txt". There is no need for a "Save As" option because the standard Java "Save" option provides that facility.

The "Close" option simply removes the currently displayed file. A dialog box enquires whether the user wishes to save the file first.

The "Print" option opens a standard print dialog that enables the user to choose the page set-up and the printer so there is no need for a "Print Set-up" option.

The editing window is of fixed size, 75 columns wide. 45 lines are printed per page, which gives an A4 printout with comfortable margins.

The editing facilities are fairly standard, including cut, copy, paste, undo, redo, find, replace, select-all, cursor movement with the arrow keys, selection of text by double-clicking, etc. In addition to these standard editing facilities, JGIDE also provides automatic completion of brackets and quotes and automatic re-indentation.

The edit menu contains eight items: Undo, Redo, Find/Replace, Re-indent, Cut, Copy, Paste, and SelectAll.

The "Undo and Redo" options work in the usual way: the number of levels of undo and redo are limited only by available memory.

Selecting the "Find/Replace" option produces a non-modal dialog, shown in Figure 3. To use the dialog to find a string in the document, enter the string and click "Find". Search proceeds from the current cursor position towards the end of the document. If found, the string is highlighted in the text. To find the next occurrence of the same string click "Find" again. If the string is not found then an appropriate message appears.

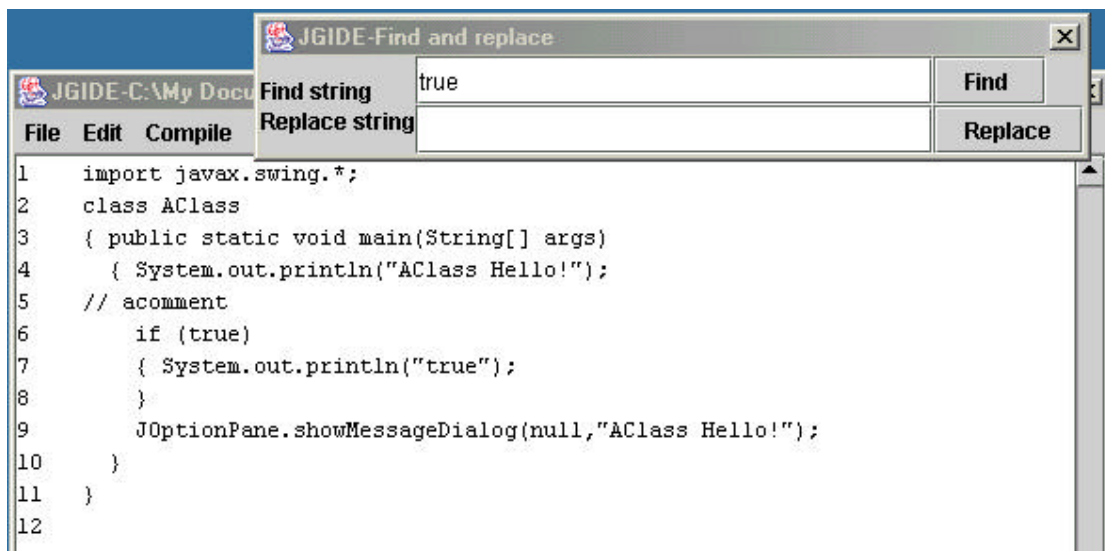


Figure 3

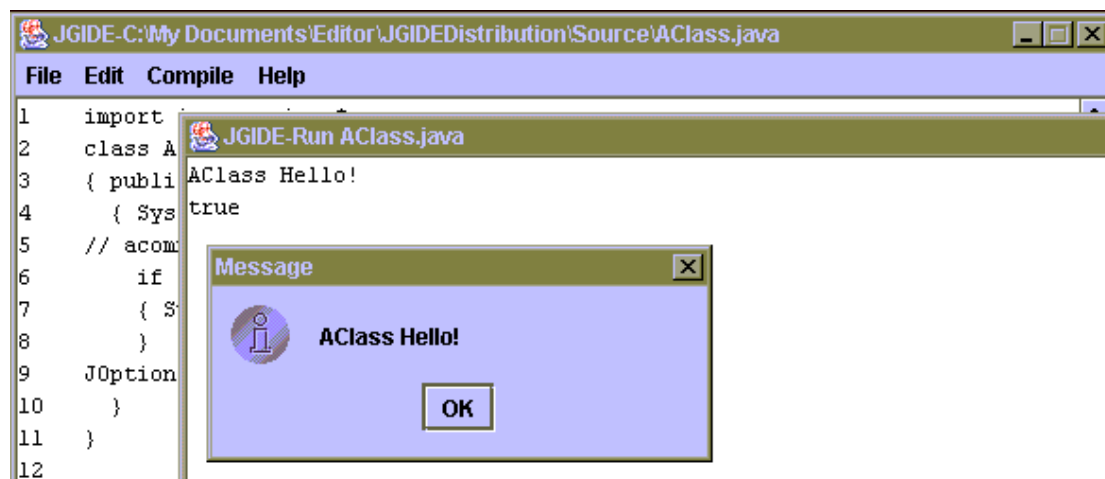
To replace a string, enter the string and its replacement in the dialog box and use "Find" as just described, and then click "Replace". Clicking "Find" and "Replace" alternately will replace successive occurrences of the "Find string" by the "Replace string". Both the find-string and the replace-string may be changed during the process: searching always continues from the current cursor position to the end of the document. This Find/Replace facility is simple but seems quite effective.

Selecting the "Re-indent" option converts the text in the window to a standard form having indentations and with curly brackets placed as shown in Figure 3. The re-indentation method implements a parser based on a simple grammar that recognises end-of-line, curly brackets, comments, other text and white space.

As can also be seen in Figure 3, JGIDE numbers the lines of text on the left. These line numbers are printed but are not stored with the text. Lines are automatically renumbered whenever you hit the "Return" key in the edit area.

The compile menu contains only two items - Compile and Run. Selecting the "Compile" option invokes the Sun javac compiler to compile the Java source code displayed in the editor window. The results of compilation are reported in a separate window, so that any errors reported can easily be compared with the source file.

Once a Java program has been compiled without errors, then selecting the "Run" option invokes the Java interpreter on the ".class" file corresponding to the ".java" file in the editor window. The output from the interpreter, including any errors, is shown in a separate window, like the compiler output. This window can be closed when no longer required. Figure 4 shows the result of running the program "Aclass.class" compiled from the source code shown in Figure 3. The "Compile" window is cleared before every compilation or run.



**Figure 4**

The help menu contains two items, "Help" and "About" that are not yet implemented.

## 5 Comparison with other IDEs

Three other Java IDEs are compared: Text Pad, JavaIDE for Windows and Jed.

TextPad is a reliable inexpensive commercial product that can be used as an IDE for several languages, including Java, C, C++ and HTML. It couples itself automatically to the Sun jdk if one is already installed on the computer. A trial version can be freely downloaded. It provides many more facilities than JGIDE and more than a student requires. It is copyrighted by Helios Software Solutions (2) and source-code is not distributed with the executable code.

JavaIDE for Windows is freely downloadable and copyrighted: source code is not distributed. It provides a set of facilities similar to that provided by JGIDE plus a few additional facilities including the ability to generate the HTML code required to run an Applet.

Jed is an interesting Java IDE. It is written in Java, both the source code and a well-illustrated description of its design and development are provided, together with some suggested

exercises for students. The design of Jed is similar in some respects to that of JGIDE. It is worthwhile to download and read the material provided on the author's web-site (4). Jed provides facilities that in this authors' view are not really required, like changing fonts, whilst providing an unsatisfactory implementation of some more essential facilities e.g. "Printed lines should quietly vanish when they reach the right margin of the paper, no wrapping". The author has copyrighted Jed. The code, about 2000 lines, is monolithic although it does use inner classes.

## **6 Discussion and conclusion**

By adopting a minimal set of requirements and using the facilities already available in the Java class library a capable application can be implemented with relatively little code.

At the time of writing, all the facilities described in this paper seem to work as intended but JGIDE has not been thoroughly tested and it has been run only under MSWindowsME.

In January 2003 you may download JGIDE from my web site (5). You may freely distribute JGIDE under the terms of the GNU General Public Licence.

If there were sufficient interest to develop a Java IDE under the GNU licence then it would be for the Java teaching community to decide whether to develop or replace the particular set of requirements, the particular design and the particular implementation (JGIDE) described here.

## **7 References**

- 1 "The Cathedral and the Bazaar", E. S. Raymond, Jan 2001, O'Reilly.
- 2 <http://www.textpad.com>
- 3 je@brighton.a.uk
- 4 [jrfisher@csupomona.edu](mailto:jrfisher@csupomona.edu)
- 5 <http://www.sbu.ac.uk/~molepd>

## **Teaching Java, XML and e-Commerce: Some reflections of past and future approaches from the “front-line” at Luton University 1999-2002**

*Tim French & Des Stephens*

Department of Computing and Information Systems

Faculty of Creative Arts and Technologies

University of Luton

Park Square Campus

Luton LU1 2JU

[Tim.french@luton.ac.uk](mailto:Tim.french@luton.ac.uk)

[Des.stephens@luton.ac.uk](mailto:Des.stephens@luton.ac.uk)

### **1. Our students and courses: a brief overview**

Our undergraduate student entry to the Computing and Information Systems Department typically come from heterogeneous experiential backgrounds in terms of their prior computing, English language skills, as well as their wider ‘world-view’ and diverse cultural backgrounds. Most of our conversion MSc students are recruited from overseas from diverse educational and cultural backgrounds and increasing numbers of overseas entrants join our undergraduate programmes, with particularly rapid growth from Mainland China. Furthermore, our own personal perception is that first year students typically join us with relatively weak ‘A’ level scores or equivalents such as GNVQ advanced (the average entry point for our undergraduates at Luton University being 8.2 for 2001-2 entry). This year’s cohort are required to have a minimum of 140 UCAS tariff points.

Luton characterises itself as an ‘access’ institution and thus we have a mission to provide vocationally relevant ‘added-value’ to students, who on average have performed relatively weakly academically to date, or have been otherwise socially/economically disadvantaged<sup>1</sup>. The challenge therefore, is to present such a diverse intake (at both BSc and conversion MSc levels) with a coherent and up to date exposure to a realistic sub-set of technologies and concepts which is both academically challenging and vocationally relevant without overwhelming either our client group or indeed ourselves! We go on to describe our approach to date and offer some reflections based on our experience gained since the autumn of 1999, and proposals for future developments. Our aim is essentially to gather feedback from staff working in similar areas and with similar client groups so as to share good practice.

Within a complex mix of some 50+ modules now offered as part of twelve or so separately named awards at both BSc and MSc levels, students encounter Java at Level 1 (semester one) as their ‘first’ programming language. This first encounter is

---

<sup>1</sup> According to HEFCE Table: “*T1a: Participation of under-represented Groups in HE: Young FT University Entrants 1998-9*”, Intake from “under-represented” backgrounds at Luton was one of the highest in the sector (ranked: 18/160) [see: <http://www.hefce.ac.uk/Learning/PerfInd/2000/extra.htm>]

subsequently reinforced by a 'further Java' skills module at Level 1 (semester 2). Our 'eCommerce techniques and technologies module' (COS74-3) is currently offered at level 3. A close variant of the 'eCommerce' module is also offered to our MSc conversion students and both these modules offers us with an exciting potential to re-examine both Java, XML as well as such derivatives languages as WML and XHTML within the generic "wrapper" of eCommerce systems deployment. One issue of concern is that for the MSc client group, they are introduced to Java, Oracle and eCommerce in parallel, rather than serially as do our BSc students. A perennial issue of concern is that we face increasing numbers of students on these modules (typically 80-350+ at BSc and 80+ at MSc levels).

## **2. Our response: Java**

### **2.1 Context**

We introduced Java as our first programming language in the academic year commencing 2000/1 and Java forms the basis from which an introduction to programming and object oriented programming are delivered to our UG and HND Computing and Information Systems cohort. This spreads across the year as two modules are taken 'back to back' by most students.

### **2.2 Content**

The content of the first semester looks at the standard programming constructs, simple data types, arrays and methods and parameters using a single class model. The students have other demands placed upon them, namely becoming familiar with our Computer Laboratories and modes of working, including adjusting to student life within an unfamiliar environment. For example students need to learning to use Blackboard as a virtual learning environment, as well as *CourseMaster* (see: <http://www.cs.nott.ac.uk/CourseMaster/>) and *Questionmark* (see: <http://www.qmark.com>) software tools as assessment mechanisms as well as adjusting to the pace and rhythms of life in Luton and at a University. The second semester introduces object orientation with encapsulation, classes and objects and packages, inheritance, polymorphism in its various aspects, abstract classes and interfaces, exception handling, streams and files, io, swing, layouts and event handling, applets and Java media framework, dynamic data structures and collection classes.

## **3. Our response: XML & eCommerce**

### **3.1 Overview**

Our Level 3 (COS74-3) module introduces students to notions of e-platforms and issues such as usability, trust, security, success factors and deployment environments, including mobile platforms. After this general introduction students encounter XML operating at the enterprise level as 'middleware'. Issues such as when and where XML might or might not be the preferred choice for data repositories are explored. Interoperability and security issues are addressed. We then actively seek to engage the students in mobile platforms (mCommerce) through WML, and now XHTML, via site building using a suitable PC based emulator. This lab-work seeks to stimulate the client group (mobile platforms are widely used by 18-25 years age groups) as well as

enabling the students to build up credible skills in mobile client side applications development. This general approach has been adopted with some slight variations between BSc and conversion Masters courses since 1999 and we now feel ready to move forward to a more mature consideration of “what else should we do now”?

### **3.2 Supporting Tools & Technologies**

Teaching has been supported by a rather ‘sparse’ set of tools so far such as IE Explorer, open-source tools such as *edit-plus* and *Nokia WAP* emulators (see: <http://www.forum.nokia.com/main/>), together with various open-source XML validators/parsers. CIS have acquired an e-commerce dedicated server to support and demonstrate server side techniques. This ‘light-touch’ approach to the field has worked thus far, but we are now considering extending the pedagogic support tools environment (e.g. via deployment of XML/RDBMS mapping tools, J2EE and XML servers).

## **4. Assessment strategy**

### **Java**

The current model adopted for the level 1 (1<sup>st</sup> and 2<sup>nd</sup> semester modules) comprises 25% continuous assessment using *CourseMaster*, this really is formative although it contributes to 25% of the summative total, a mid semester *QuestionMark* test of 25% and an end of semester *QuestionMark* test of 50%. The following year we will be replacing *QuestionMark* with an alternative tool.

### **eCommerce**

Students are asked to complete two in-course assignments. Assignment one requires them to complete a detailed expert heuristic evaluation of some popular eFinance sites. For MSc students we also require a comparison to be made between mobile and conventional PC based user experiences. The second assignment requires the students to design, code and test a small-scale mobile application using XHTML-basic. A formal written three-hour examination covering the whole course includes a case study that forms the terminal (50%) assessment point of the module.

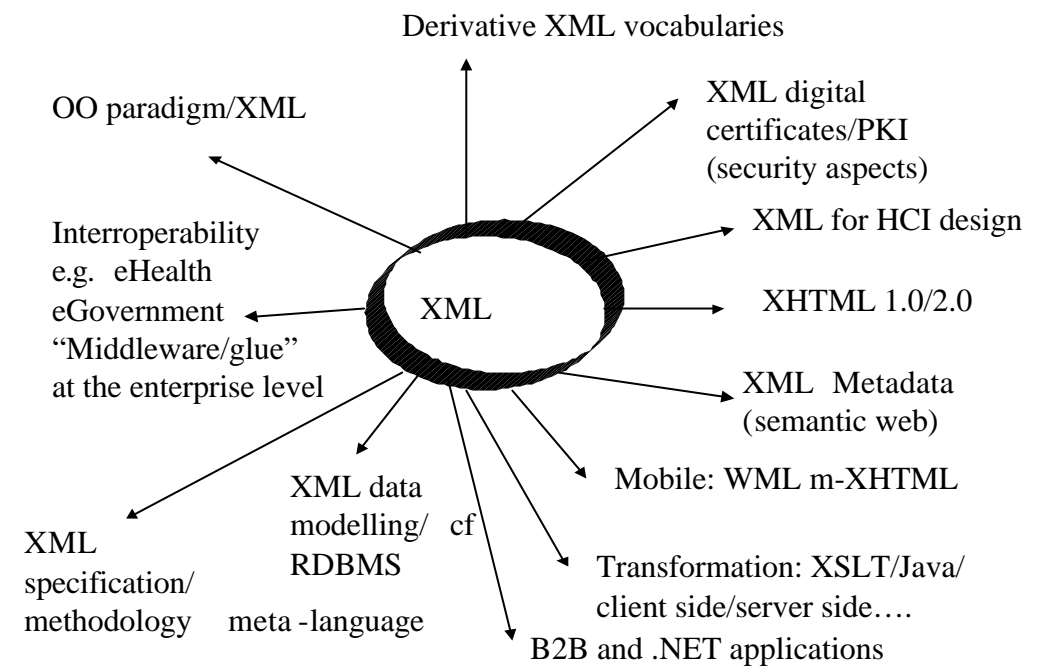
In general our students seem to have responded well to engaging in eCommerce initially from the ‘user-experience’ perspective. Consideration of strategic and data-architectures (such as XML) moves them “in” from user to product designer/procurement and finally as “coders” they experience building a small mobile application using XHTML-basic and contrasted with WML. Our external examiner has complimented us on the diversity of the student experience and on the rigorous nature of our examination paper in which students typically: answer short answer questions, correct code fragments and respond to a detailed ‘XML’ oriented case study.

## 5) **Future directions: developing an XML/Java and web centric curriculum?**

We are now actively considering creating a whole variety of new “XML centric” awards/modules to try to track some of the major developments this area has seen since 1999. We are keen now to extend our coverage of Java/J2EE and XML too. Figure 1 (which follows) illustrates the diversity and increasingly pervasive nature and influence of ‘XML’ based technologies. This presents educators with a classical dilemma: should we wait until the technology ‘matures’ or should we become an early adopter, as indeed we did back in 1999? Waiting for so-called ‘maturity’ might directly disadvantage our students through failing to provide them with credible leading edge skills sets. We have chosen thus far to run after a fast moving target, however perhaps it is now timely to reflect on where this chase has led us pedagogically. We obviously prefer to think that our chase has not led us up a blind alley but we are open to suggestions to the contrary! Our rationale for adopting an XML/Java centric computing curriculum that seeks to place these technologies at the centre stage through early adoption is briefly as follows:

- Client groups are introduced to key vocational enabling technologies through one coherent set of technologies which they can relate to easily and which are in demand vocationally;
- The increasing scope and influence of these technologies provides a wide central base from which most key ‘computing’ areas likely to be of direct relevance to our intake group can be explored both intellectually and practically. [NB: few of our intake group(s) are likely to be seeking specialist careers in areas such as: neural networks, computability and logic, formal methods or other ‘pure’ Computer Science areas];
- By exposing our client group to these directly ‘useful’ technologies in the context of eCommerce implementation we can directly fulfil our mission as an access institution: by adding value in explicit and measurable ways, whilst not of course seeking to neglect creating routes for those able to pursue specialist Master’s or research degrees. Indeed, many leading edge computing research areas can be addressed through exposure to these areas (e.g. the semantic web);
- In many senses these technologies are increasingly challenging fundamental notions (such as the role of relational data modelling and databases) and leading to rapid advances across wide areas of computing. Examples include content management (server side XML transformations using Java et al); security (XML digital certificates) and XML enabled agents. By centring the curriculum around XML/Java in all its richness and diversity we believe that we are encouraging students to innovate (and perhaps encouraging their educators to innovate their own thinking too!) not merely follow a ‘standard’ or ‘blue-sky’ computing curriculum diet;
- Centring on a sub-set of technologies acts as a necessary ‘self-limiter’. This is both desirable and necessary in an increasingly overloaded computing ‘world’. Equally, students tracking fast-moving areas soon realise the need for continuous professional development and the need for themselves to identify transferable skills.

**Figure 1: XML – a conceptual overview of some possible pedagogic sub-areas**



### 5.1 Java at level X: diversity or depth?

Our second and 3<sup>rd</sup> year students are exposed to a variety of languages, primarily C++, Prolog and Python. However, it does not seem that the potential of students is fully explored by offering diversity in place of depth. The riches of Java and of complimentary technologies should, together with initiatives such as *.NET* (see: <http://www.microsoft.com/net/>) and *Visual Studio*, be further developed as part of the core curriculum. This naturally leads to consideration of the introduction of an award such as ‘*Computer Science and Web Service Technologies*’. Such an approach is under consideration.

### 5.2 XML and eCommerce: client or server side emphasis?

One immediate option is to update the practical work from WML to mobile XHTML (essentially minor re-working) with further investigations of J2ME, whilst another idea is to explore links between J2EE (as a server) and XHTML (as client) so as to enable client side mark-up to be instantiated with simple datasets from a server. Of course yet another more radical approach might be to purchase a dedicated native XML server environment and integrate with our existing eCommerce server. There is also a potential need for us to also explore the complex interrelationships between XML, its derivatives, Java, J2EE and indeed *.NET*. We feel instinctively that not to expose our client group to some sub-set of these technologies would be to deny them key vocational skills. Our approach does offer our students a conceptual introduction to the XML area (and critically offers some answers as to “why” this technology is so critical to e-commerce deployment) and our current approach also gives students some skills in practical mobile client side scripting (WML/XHTML). We feel nervous

however as to exactly how to move forward – one problem is that the whole XML area has itself seen rapid expansion since 1999, as we allude to in Figure 1 above.

## 6. Reflections

We feel that others engaged in this domain of interest are likely to be experiencing similar dilemmas and compromises in relation to both Java and XML deployment within HE pedagogic settings. Our present situation has really been the result of a mix of serendipity, and various constraints and we are conscious that we need to share others experiences. We would be particularly keen to learn how others are seeking to address some or indeed all of the dimensions of XML as presented in Figure 1. Our assumption is that we cannot afford to ignore such a pervasive area, even if our coverage has up till now been somewhat sparse. This is particularly relevant now with the potential introduction of a wide range of “top-up fees” leading to a further segmentation of the HE market. One segment being *vocational relevance* as an important motivation to students alongside low-end fees, particularly given the rise in ‘Corporate Universities’ as competitor HE institutions see: Paton R. and Taylor, S. (2002). We may indeed begin to see increased demands from all HE segments (elite, access, open, corporate models) for curricula which are more strongly vocationally oriented. In this context the role of XML, eCommerce and Java as enabling technologies needs to be continuously re-examined. By deliberately self-limiting the core of our curriculum to such an overtly vocational and fast moving area we feel it may be time to reflect on the likely implications and impact:

- What types of core student learning are taking place when we choose to self-limit the range of technologies we expose to our students? What are our students ‘missing out’ on if anything conceptually? We believe we can demonstrate that an XML/Java based curriculum teaches ‘core’ universal computing concepts;
- What are the practicalities involved in ‘tracking’ fast moving vocationally oriented technologies in such an explicit manner? Clearly there are resource, staff training and other similar implications involved. There are also dangers too – how sure are we that such areas will be so pervasive in 5+ yrs time? Will the necessary funding and institutional support systems prove adequate to support continual innovation?
- More fundamentally: are we ‘dumbing down’ the computing curriculum or simply attempting to partially emulate the Corporate University vision of HE by creating an explicit vocational skill and conceptual delivery system matched to our client groups (i.e. our students and their employers)?

We trust these and other related issues will be explored in a lively discussion following our oral presentation.

## References

Paton, R. and Taylor (2002) ‘*Corporate Universities – historical development, conceptual analysis and relations with public-section Higher Education*’. Report published by the Observatory on Borderless Education, [available upon request from [F.M.Leslie@open.ac.uk](mailto:F.M.Leslie@open.ac.uk), Open University Media Relations Department.]

## **Pedagogic teaching resources (selected)**

- **Student Texts:**

Horstmann, C (2002) *'Big Java'*, John Wiley,

Carey, K, Stanko, B, (2002) *'XML Content and Data'*, Prentice Hall.

Castro, Elizabeth (2001) *'XML for the world wide web'*, Visual Quickstart Guide, Peachpit Press, USA.

Dietel, Dietel, Nieto, Steinbuhler, (2002) *'Wireless Internet & Mobile Business - How to program.'* Prentice Hall, UK.

Bandyo-padhay, N. (2002) *'e-Commerce context, concepts and consequences'*, McGraw-Hill, UK.

- **Some frequently used W3 sites:**

[www.useit.com](http://www.useit.com) [Nielsen , J. home site: user experience, last accessed 19-10-2002]

[www.xml.com](http://www.xml.com) [Useful source of up to date news, resources, technical materials, last accessed 19-10-2002]

[www.smdf.org](http://www.smdf.org) [French *et al* research materials, semiotics of e-commerce, last accessed 19-10-2002]

---

## Web Services and Java

*Chrisina Draganova*

Department of Computing, Communication Technology and Mathematics

London Metropolitan University

100 Minories, London EC3 1JY

Email: [c.draganova@londonmet.ac.uk](mailto:c.draganova@londonmet.ac.uk)

### Abstract

This paper gives an overview of Web Services and the latest Java technologies developed in this respect. It also gives some ideas of how to include these new technologies in the computing curriculum.

## 1. Web Services

### 1.1 Introduction

Web Services are called any services that are available on the Web. Web Services can be published, discovered and invoked over the web. Web Services can be implemented in any available technology but they are accessible through a standard protocol. The main idea is to use the Web not only for accessing information but for accessing services as well. The services could be considered as reusable components used for building bigger services, which communicate with each other in a way independent of any programming languages and platforms.

Examples of popular Web Services are: a stock quotes service, a service for finding the most-effective delivery route, credit card service, currency conversion, language translation, search service, etc.

Many organizations work on developing standards and technologies for Web Services. The World Wide Web Consortium (W3C) [16] and The Organization for the Advancement of Structured Information Standards (OASIS) [9] have different activity groups working on Web services standards, architecture and technologies for Web Services design. The Web Services Interoperability Organization is another open industry initiative devoted to promoting Web Services interoperability across platforms [14]. Some of the major Web Services platforms available at present are Sun Microsystems Inc. Java™ Web Services Developer Pack [8], Microsoft .NET [11], IBM WebSphere SDK for Web Services (WSDK) [4] and Oracle® support for Web Services [19].

The main purpose of using Web Services is to provide interoperability between heterogeneous platforms.

The basic requirements necessary to make Web Services work are:

- discovering a location of a service provider
- discovering provided services
- providing a way of communication with a particular service
- providing a way to execute available functions
- providing a standard messaging
- providing a way of data representation

Some books describing Web Services in detail include [1], [2] and [17].

### 1.2 Web Services standards

The Web Services standards are based on Extensible Markup Language (XML) used for describing and exchanging data. XML is a markup language such as HTML but with flexibility of defining new tags. The current XML standards are [3]:

- Strict syntax rules for creating XML documents
- XML Schema and XML Document Type Definition (DTD) standards for the validation of XML documents and specification of XML types
- XML Namespaces for combining multiple sources in a single document
- XML processing providing mechanisms for creating, parsing and manipulating XML documents

Appendix 5.1 shows an example of a XML document.

The main Web Services standards comprise:

#### Simple Object Access Protocol (SOAP)

SOAP is a lightweight, XML-based protocol including three parts: an envelope with description of what is in the message and how to process it, encoding rules for data types and mechanisms for remote procedures calls and

responses. SOAP messages can be exchanged using different Internet protocols as HTTP, SMTP and FTP. Other protocols as RMI (Remote Method Invocation) and JMS (Java Message Services) are also possible, although HTTP is the most widely used.

The SOAP specification defines the structure of a SOAP message, which is an XML document with a root element **Envelope** and sub-elements:

- **Header** – optional element containing information about the message, routing, delivery and security information
- **Body** – mandatory element containing data or instructions intended for the recipient
- **Fault** – optional element containing error information

The SOAP Envelope element contains the name spaces and schema information for the message. The SOAP encoding provides support for simple data types as integer, float, string, etc., as well as for compound types as structs, arrays, etc.

SOAP supports two types of information exchange: Remote Procedure Call (RPC) exchange and document-oriented exchange. With the RPC exchange the SOAP message body specifies the Web service method to be invoked, the parameters that this method takes and the Universal Resource Identifier (URI) of the target procedure. After the call is executed, the Web Service can send back to the calling application a SOAP message with the results of the method call. With the document-oriented SOAP messages no method is invoked and they are used for notification or status information. The SOAP messages can be transmitted synchronously or asynchronously depending on the requirements of the specific application.

A SOAP message may include attachment parts containing any kind of content: image file, plain text, XML data document, etc.

Appendix 5.2 shows an example of simple SOAP message.

The current specification of SOAP could be found on the W3C web site [12]. Some of the existing implementations of SOAP are: Apache Software Foundation [13], GLUE [10], and Microsoft's .NET [11]

### **Web Service Description Language (WSDL)**

WSDL is an XML format standard for describing the interface of a web service. The WSDL description gives information about what exactly a web service does, how to invoke its functions and where to find it. A WSDL document has an element called **definitions** that contains the following sub-elements:

- **portType** – description of the interface of a Web service; i.e., what the Web service does
- **operation** – definition of a method signature on a Web service
- **message** – an optional element within an operation, used as input message, output message or a fault message
- **types** – information about user-defined XML types.
- **binding** – description how to invoke operations using specific messaging/transport protocol
- **port** – specifies the network address of the endpoint hosting the Web service.
- **service** – specifies the ports related to the same service interface (**portType**) but using different protocols (bindings)

Appendix 5.3 shows an example of a WSDL document describing a service called “Hello”.

Detailed description of WSDL can be found in [16] and [20].

### **Universal Description, Discovery, and Integration (UDDI)**

This is a registry standard, which allows organisations to publish and discover Web Services using standardised methods. They are often compared with white, yellow and green pages, because they contain contact, categorisation and technical information. The UDDI is an industry initiative to provide a platform-independent framework for creating a UDDI Business Registry. There are currently several providers of UDDI registers called UDDI Operators. For example IBM and Microsoft host such registers. They also offer free public Web-based interfaces for registering and finding businesses. The information in these registers is replicated. Appendix 5.4 shows a search made for service “Hello” on the IBM UDDI Web Browser Interface: <https://uddi.ibm.com/ubr/registry.html>.

The UDDI specification defines a set of data structures and an Application Programming Interface (API) for registering and finding businesses. The UDDI specification also allows organisations to create their own UDDI registries in order to have more control for the access and the updating of information, and the reliability of the registry content.

The main data structures are:

- **businessEntity** – the top-level XML element that provides contact, identifier, and taxonomy information
- **businessService** – a child of a **businessEntity** element and a parent of a **bindingTemplate** element that gives business service description, technical service description and category information.
- **bindingTemplate** – provides information how to invoke or bind to a specific Web Service
- **tModel** – provides technical specifications of the Web Services in order to provide standards and templates for different implementations of identical Web Services

The UDDI specification provides two types of API operations

- Publication API – support set of operations that allows organisations to publish and update information to the UDDI registry
- Inquiry API – support set of operations that allows users to extract information out of the UDDI registry

Corresponding to the four data structure types (**businessEntity**, **businessService**, **binfingTemplate** and **tModel**) UDDI specification provides operations for save, delete, find and get details operations. They can be grouped in the following table [2]

	Business	Service	Binding	tModel
Save/Update	save_business	save_service	save_binding	save_tModel
Delete	delete_business	delete_service	delete_binding	delete_tModel
Find	find_business	find_service	find_binding	find_tModel
GetDetail	get_businessDeta il	get_serviceDetai l	get_bindingDetai l	get_tModelDeta il

The latest specification of UDDI is available on [15].

## 2. Related Java technologies

One of the present Java based platforms for developing Web services is the Java™ Web Services Developer Pack (Java WSDP) [8]. It provides Java standard implementations of the main Web Services standards including SOAP, WSDL and UDDI. It supports also Java standard implementations for Web application development such as JavaServer Pages™ (JSP™ pages) and the JSP Standard Tag Library.

The latest version of Java WSDP includes the following APIs related to Web Services [8]:

- Java™ Architecture for XML Binding (JAXB)
- Java™ API for XML Messaging (JAXM)
- Java™ API for XML Processing (JAXP)
- Java™ API for XML Registries (JAXR)
- Java™ API for XML-based Remote Procedure Call (JAX-RPC)

The Java™ Web Services Tutorial provides an easy guide for using the above APIs [7]. Comprehensive description of these APIs could be found in [18]. More in depth explanation of Java Web Services with examples is given in [5] and [6].

**2.1 JAXB** gives an efficient and standard way of mapping between XML documents and Java objects. Given a DTD and a binding schema, JAXB compiler generates Java classes. As and XML document is an instance of a schema, a Java object is an instance of a class. JAXB allows converting existing XML data into Java objects and creating new XML data from Java objects.

**2.2 JAXM** enables applications to send and receive document oriented XML messages using a pure Java API. JAXM implements Simple Object Access Protocol (SOAP) 1.1 with Attachments messaging. JAXM supports asynchronous messaging, routing of message to more than one party and guaranteed delivery messaging. The complete JAXM API includes two packages:

- **javax.xml.soap** – provides SOAP messaging APIs for sending request-response messages
- **javax.xml.messaging** – provides APIs for using a message provider and sending one-way messages

**2.3 JAXP** supports processing of XML documents using DOM (Document Object Model), SAX (Simple API for XML Parsing) and XSLT (XML Stylesheet Language Transformations). The packages of the JAXP APIs are

- **javax.xml.parsers** – provides a common interface for SAX and DOM parsers
- **org.w3c.dom** – defines the Document class and all other classes of the DOM components
- **org.xml.sax** – defines the SAX APIs
- **javax.xml.transform** – defines the XSLT APIs

The processing of XML documents using SAX is element-by-element while with DOM the entire XML structure is loaded in the memory.

**2.4 JAXR** provides a uniform and standard Java API for accessing different kinds of XML Registries and registering a business. JAXR has two main packages:

- **javax.xml.registry** – provides API interfaces and classes for defining the registry access interface
- **javax.xml.registry.infomodel** – provides interfaces for defining the types of objects that reside in a registry and the way they relate to each other.

Many methods in the JAXR API use a Collection object as an argument or a returned value allowing operations on several registry objects at a time.

**2.5 JAX-RPC** provides the core API for developing and deploying Web Services on Java platform. JAX-RPC supports SOAP and WSDL standards for cross-platform interoperability. JAX-RPC contains the following packages:

- **javax.xml.rpc** – provides the core JAX-RPC APIs for the client side
- **javax.xml.rpc.encoding** – provides interfaces for XML processing implemented by the JAX-RPC runtime system, interfaces for converting XML representation to a Java object and the opposite, and interfaces for the representation of a type mapping
- **javax.xml.rpc.handler** – provides APIs for SOAP Message Handlers
- **javax.xml.rpc.handler.soap** – provides an interface for access to the SOAP message for either RPC request or response
- **javax.xml.rpc.holders** – contains the standard Java Holder classes as BigDecimal, BigInteger, etc.
- **javax.xml.rpc.server** – provides APIs for the service based JAX-RPC endpoint
- **javax.xml.rpc.soap** – provides the SOAPFaultException class that represents a SOAP fault

JAX-RPC uses the concepts of “stub” and “tie”. The client application making the remote call has a local object “stub” representing the remote service. The remote method call is converted into a SOAP message and then transmitted as an HTTP request. After the server receives the HTTP request, the SOAP message is extracted from the request and translated into a method call. The server has a “tie” or “skeleton” object representing the service that is responsible for reconstructing the data and invoking the actual method with the expected parameters and data types. The result of the method execution is processed in a similar way. JAX-RPC provides also APIs for invoking Web Services dynamically through the Dynamic Invocation Interface (DII) using dynamic proxies – objects generated during runtime. The information needed for creating DII client is provided by the WSDL document of the Web Service.

### **3. Web Services in the computing curriculum**

In teaching computing we have to emphasise principles as well as demonstrate them through specific technologies. Teaching Web Services is an excellent way to give our students knowledge about leading technologies and in the same time demonstrate fundamental computing concepts.

#### **3.1 Technologies related to Web Services in the current computing curriculum**

Some of the technologies related to Web Services such as XML, RPC and JSP are already included in several units taught at the Department of Computing, Communication Technology and Mathematics, London Metropolitan University. These are:

- Introduction to Web Development QC213 (BSc Computing second year unit) - JSP, XML
- Programming Solutions for the Internet QC310 (BSc Computing third year unit) -JSP, XML and Web Services Frameworks J2EE and .NET
- E-Commerce (II) QB316 (BSc Business Information Technology third year unit) - XML
- Web site Design & HTML QH208 (HND Computing second year unit) – XML
- Web Design and Implementation QF108 (Foundation Degree in Computing first year unit) - XML

Introducing topics on Java Web Services is expected to be perceived well by computing students in advanced programming units, since Java is the main programming language that they study. This will provide opportunities for the students to:

- reinforce the learning of fundamental programming concepts
- understand better the present Internet technologies
- understand interoperability issues in distributed computing
- understand business issues related to Web Services
- understand the importance of using open standards

- acquire knowledge of leading technologies
- gain marketable skills

At present Web Services topics are introduced as mentioned already in the BSc Computing third year unit “Programming Solutions for the Internet”. The students are familiarised with two Web Services Frameworks J2EE and .NET. They are expected also to develop skills in implementing Web Services in Java using Oracle JDeveloper 9i.

Final Year Projects on developing Web Services for a particular business application is another example of including these technologies in the computing curriculum.

### 3.2 Teaching more on Web Services

We propose sample sets of aims, learning outcomes and topics of study for developing two independent units on Web Services. The first unit to be designed is for students who are interested more in the business side of Web services. The second unit would be for students interested in the technical implementation of Web Services.

#### Web Services - business unit

##### Aims:

- To provide students with understanding of key issues related to Web Services and the underlying technologies
- To develop a critical analysis for evaluation of business needs for developing and using Web Services
- To develop awareness of existing platforms related to Web Services

**Learning Outcomes:** On completion of this unit, students will be able to:

- Understand what a Web Service is, its purpose and use
- Understand the underlying technologies to Web Services
- Identify the main issues in applying Web Services to specific business
- Analyse and specify appropriate solutions for Web Services applications
- Evaluate different tools and platforms for building and using Web services

##### Topics of Study:

- Web Services: introduction, history, evolution
- Web Services standards: XML, SOAP, WSDL, UDDI
- Web Services Business Models
- Evaluation of existing Web Services Platforms
- Web Services in the market
- Web Services Security issues
- Case Studies

#### Web Services - computing unit

##### Aims:

- To provide students with deep understanding of Web Services standards and technologies
- To equip students with technical skills for implementing Web Services
- To develop a critical analysis for evaluation of different platforms for implementing Web Services

**Learning Outcomes:** On completion of this unit, students will be able to:

- Understand what a Web Service is, its purpose and use
- Understand the underlying Web Services standards and technologies
- Understand the requirements for interoperability
- Understand how to implement, publish, find and invoke a Web Service using a specific platform
- Evaluate different tools and platforms for building and using Web services

##### Topics of Study:

- Web Services: introduction, evolution of distributed computing, standards
- Data portability: XML, Document Type Definitions and Schemas, Namespaces
- Protocols: SOAP: architecture and implementation
- Describing Web services: WSDL
- Publishing and discovering Web Services: UDDI
- Evaluation of existing Web Services Platforms
- Java Web Services technologies and implementation

- Java APIs: JAXP, JAXB, JAXM, JAXR, JAX-RPC
- Step by Step Examples
- Web Services Security issues and implementation

#### 4. Conclusions

Web Services provide potential for extending the Web capabilities to provide interoperability between computer applications written in different programming languages and implemented on different platforms. The Web Services technologies are based on open standards promoting collaboration between organisations. Most of the major computing companies and organisations contribute to the development of the standards and tools for creating Web Services. There is no doubt that this is an important step in the computing field and the computing curriculum should include this new development. Teaching topics on Web Services more extensively would give the students exposure to leading technologies, deeper understanding of fundamental concepts and developing valuable marketable skills.

#### 5. Appendices

##### 5.1 Example of a XML document

```
<?xml version="1.0"?>
<!--Example of a XML document-->
<staff>
  <lecturer id="1">
    <surname>Draganova</surname>
    <firstname>Chrisina</firstname>
    <address>
      <number>100</number>
      <street>Minories</street>
      <city>London</city>
      <postcode>EC3 1JY</postcode>
    </address>
  </lecturer>
</staff>
```

##### 5.2 Example of a SOAP message

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Example of a SOAP message-->
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope">
  <SOAP-ENV:Header>
    <ns1:username xmlns:ns1="HelloService">myusername</ns1:username>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:getHello xmlns:ns1="HelloService">
      <name xsi:type="xsd:string">MyName</name>
    </ns1:getHello>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

##### 5.3 Example of a WSDL document

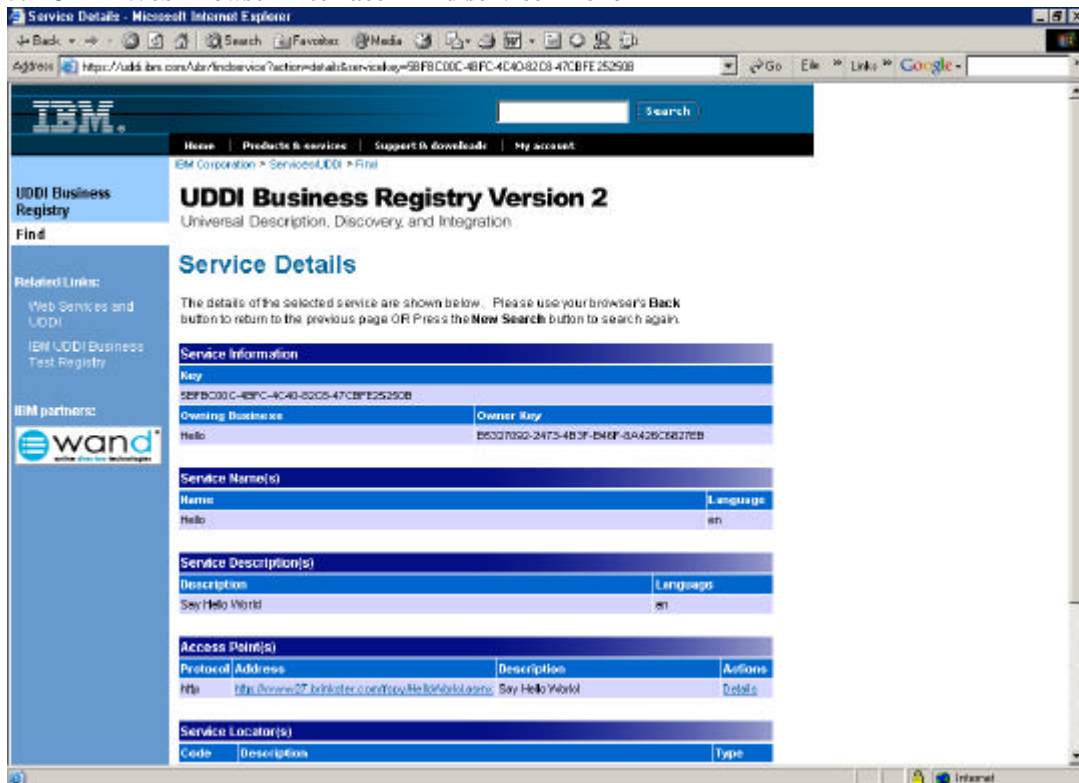
```
<?xml version="1.0">
<!--Example of a WSDL document-->
<wsdl:definitions
  name="HelloImpl"
  targetNamespace="http://localhost.com/services/Hello"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://localhost.com/services/Hello"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <wsdl:message name="getHelloInput">
```

```

    < wsdl:part name="name" type="xsd:string"/>
</ wsdl:message>
< wsdl:message name="getHelloOutput">
    < wsdl:part name="output" type="xsd:string"/>
</ wsdl:message>
< wsdl:portType name="HelloImpl">
    < wsdl:operation name="getHello" parameterOrder="p0">
        <wsdl:input name="getHello" message="tns:getHelloInput"/>
        <wsdl:output name="getHello" message="tns:getHelloOutput"/>
    </ wsdl:operation>
</ wsdl:portType>
<wsdl:binding name="getHelloImplSOAPBinding0" type="tns:HelloImpl">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
style="rpc"/>
    <wsdl:operation name="getHello">
        <soap:operation soapAction="" style="rpc"/>
        <wsdl:input name="getHello">
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:localhost.com.services.Hello"/>
        </wsdl:input>
        <wsdl:output name="getHello">
            <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="urn:localhost.com.services.Hello"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="Hello">
    <wsdl:port name="Hello" binding="tns:HelloImplSOAPBinding0">
        <soap:address location="http://localhost.com/services/Hello/">
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

#### 5.4 UDDI Web Browser Interface – find service “Hello”



## 6. References

1. A Web Service Primer, Vasudevan V  
<http://www.xml.com/lpt/a/2001/04/04/webservices/index.html>
2. Building Web Services with Java™ : Making Sence of XML, SOAP, WSDL and UDDI, S Graham, Simeonov S, Boubez T, Davis D, Daniels G, Nakamura Y, Neyama R, Sams Publishing 2002
3. Extensible Markup Language, W3C Architecture Domain  
<http://www.w3.org/XML>
4. IBM WebSphere SDK for Web Services (WSDK)  
<http://www-106.ibm.com/developerworks/webservices/wsdk/>
5. Java™ and SOAP, Englander R, O'Reily&Associates, 2002
6. Java™ Web Services, David A., Cbappell & Tyler Jewell, O'Reily&Associates, 2002
7. Java Web Services Tutorial, Sun Microsystems, Inc. <http://java.sun.com/webservices/docs/1.0/tutorial/index.html>
8. Java WSDP, Sun Microsystems, Inc.  
<http://java.sun.com/webservices/webservicespack.html>
9. Organization for the Advancement of Structured Information Standards  
<http://www.oasis-open.org/>
10. SOAP implementation : GLUE  
<http://www.theminelectric.com/products/glue/glue.html>
11. SOAP implementation: Microsoft .NET  
<http://msdn.microsoft.com/webservices/default.aspx>
12. SOAP Version 1.2, W3C Working Draft 26 June 2002 <http://www.w3.org/TR/2002/WD-soap12-part1-20020626/>
13. The Apache XML project: Apache SOAP, <http://xml.apache.org/soap/index.html>
14. The Web Services Interoperability Organization , <http://ws-i.org/AboutUS.aspx>
15. UDDI organisation web site, <http://www.uddi.org>
16. Web Services Activity, W3C Architecture Domain, <http://www.w3.org/2002/ws/>
17. Web Services Essentials, Gerami E, O'Reily&Associates., 2002
18. Web Services Made Easier: The Java™ APIs and Architectures for XML, A Technical White Paper, Sun Microsystems, Inc., <http://java.sun.com/xml/webservices.pdf>
19. Web Services Center, <http://otn.oracle.com/tech/webservices/content.html>
20. WSDL 1.1 W3C Note 15 May 2001, <http://www.w3.org/TR/wSDL>

# Designing Reusable Java Teaching Modules

Safia Barikzai

School of Information Technology and Computer Science,  
Faculty of Informatics,  
University of Wollongong,  
NSW, Australia

**Email:** safia@uow.edu.au

## Abstract

Java is fast becoming the most used language in the IT industry today. It is also the most popular language for teaching software development/programming in most Higher Education institutions across the world. As academics, we try to instill the values of good software development principles such as reuse, code sharing, platform independence, and so on to our students. We not only teach our students about the programming language i.e. Java but also place emphasis on good durable principles of software development. However, do we as academics practice what we preach when it comes to delivering our lectures and tutorials? Do we produce teaching resources that are easily accessible to our students and colleagues alike (cross platform)? How reusable are these resources?

Java technology is dubbed '*write once, run anywhere*'. The world of education is slowly gearing towards the notion of 'design and develop once, study anywhere, anytime and anyplace'. This paper is a '*work in progress*' and looks at various research developments emerging in the educational technology domain in order to find an educational equivalent to '*write once, run anywhere*'.

## Keywords

Java, Multimedia Programming, Reusable Teaching Components, Learning Design

## 1. Introduction / Background

These days, students are required to have Java language skills in a variety of courses and subjects within the Information Technology and Computer Science curriculum, especially at

postgraduate level. At postgraduate level, it is quite common to have a class of students with varying skill sets including some that have very little programming experience.

Multimedia Programming Foundation is a new core subject offered within the new Master of Multimedia course at the School of Information Technology and Computer Science (SITACS) [1]. The subject will also be available as an elective to other Masters courses within the school. This subject will no doubt have a Java teaching component. The learning activities have to be tailored such that students with varying skill sets can accomplish the same Learning Outcomes.

## 2. Student Skills / Experiences

It is important to identify potential student types before setting off to design a new teaching module. This is not an easy task, as the student skill sets will vary depending on which course they enrolled on. The different student skill sets are identified in order to tailor the activities and tasks for each skill set. (see table 1)

Table 1 - Student Skills / Experiences							
Type	Undergraduate Degree	Computing Experience	Analysis and Design		Imperative Programming e.g. Ada, C, VB, Delphi	OO Programming	
			SSADM or Yourdon	UML or OMT		C++	Java
1	Non IT	N	N	N	N	N	N
2	Business IT	Y	Y	N	Y	N	N
3	Computing	Y		Y	N	Y	N
4	Computing	Y		Y	Y	Y	Y
5	Computing	Y		N			Y
6	Computing	....	....	....	....	....	....
7	Computing	....	....	....	....	....	....
8	Computing	Y	Y	Y	Y	Y	Y

Some initial thoughts on the student types:

1. No students will be of Type 1.

Most students doing this subject will have computing background and therefore will be familiar with computing concepts.

2. A majority of the students will be of Type 2, 3 and 4.

These students will be able to do analysis and design activities and would have encountered programming in some shape or form but not necessarily encountered Objected Orientation or Object Oriented Programming.

3. A few will be of type 5.  
These students have some Java language skills but have not done any real Object Oriented Analysis and Design.
4. A small minority will be of Type 8.  
These students will have both Object Oriented analysis and design skills as well as familiarity with the Java language.

### 3. Java Teaching Component

#### *Overview*

While designing the Java teaching component for the Multimedia Programming Foundation subject, different ways of maximizing the reusability of the resource will be looked at. This includes making use of freely available on-line tutorials from reputable sources such as IBM, Sun and Apple. Ideally the Java teaching component should be easily incorporated into other subjects offered within the School. For example, an ‘eBusiness Programming Foundation’ subject should easily be able to ‘reuse’ the Java teaching component.

The *proposed* Java teaching component content is outlined in table 2. A session includes a one-hour lecture, one-hour interactive presentation followed by an hour lab activity. Students are required to spend at least another 6 hours of their own time doing the exercises. Each session is self-contained with a set of learning outcomes and goals. Each of these sessions also includes tailored activities for the student types identified (see table 1).

Session 1. Meet the development environment Java – review of language basics	(Programming oriented) (Programming oriented)
Session 2. Review of OO / classes / frameworks – theory Review of OO / classes / frameworks – Java	(Programming oriented) (Programming oriented)
Session 3. MM Frameworks (theory)	(MM oriented)
Session 4/5 Constructing applications      Swing (GUI)	(Programming oriented)

If the above were to be used in the 'eBusiness Programming Foundations' subject then Session 3 - 'Multimedia Frameworks' can be replaced by 'eBusiness Frameworks'. Resource sharing can be achieved both at the session level as well as at the subject or module levels. Some resources naturally can only be reused within the setup of the School where the subject is being taught. For example, 'Session 1 - Meet the development environment' is specific to our School.

### **Java Development Environment**

Sun Solaris, Windows and Apple Mac OS X v10.2 platforms are used to teach students software development at SITACS. The computer laboratories have a wide selection of development tools available for staff to teach with and students to learn. These tools vary from the very simple tools such as TextPad [2] and BBEdit [3] (on the Macintosh platform) to using sophisticated environments such as IBM WebSphere Studio Application Developer [4]

SITACS is a member of the IBM Scholar Program [5] whereby we can freely use IBM's J2EE eBusiness integrated development environment, WebSphere Studio Application Developer. WebSphere Studio Application Developer is built on the Eclipse[6] open source integrated development environment. The Apple Mac OS X v10.2 environment has Cocoa application environment [7] and Apple WebObjects [8] for the learning and teaching of Java and Multimedia programming. Subjects within the eBusiness stream [9] make extensive use of the WebSphere environment whereas Cocoa and WebObjects are used as the Java IDEs in the Multimedia stream.

Given that Java is designed to be a portable language, moving Java code from one environment to another is reasonably straightforward. By comparison moving the educational resources from one stream to another is far more problematic.

## **4. Learning Designs**

It is assumed that most academics design educationally sound learning resources. However, often these resources are not available for 'sharing' amongst colleagues. There could be a number of factors contributing to this. The most common one being lack of promotion and interoperability of an existing resource.

*“The development of a framework that supports pedagogical diversity and innovation, while promoting the exchange and interoperability of e-learning materials, is one of the key challenges in the e-learning industry today.”* IMS Learning Design Best Practices and Implementation Guide [10].

The aim is to make use of the IMS Learning Design specifications in order to enhance reusability of the Java resources. Some of the Key points from the IMS Learning Design specifications are given in table 3.

**Table 3 - IMS Learning Design**

The Learning Design specification needs to:

- describe and implement different kinds of learning approaches
- enable repeatable, effective, and efficient units of learning
- provide access to, and interchange of, units of learning between learning systems
- support multiple delivery models
- support reuse and re-purposing of units of learning or their component elements
- support the reuse or repurposing of the framework and components of a unit of learning
- support multiple learners and multiple roles in a learning activity, reflecting learning experiences that are collaborative or group-based

## **5. Summary**

This paper is a *‘work in progress’*. There are more questions than answers. The idea of creating reusable teaching and learning resources sounds good but is this viable? The idea of IMS Learning Design specifications in order to achieve reusability of educational resources is attractive but is it too complex in practice? Are IMS Learning Design specifications compatible with approaches to teaching Java? What experiences do other Java educators at the conference have of specifically formulating reusable resources?

## **References**

- [1] School of Information Technology & Computer Science,  
University of Wollongong, Australia  
<http://www.sitacs.uow.edu.au>

- [2] TextPad Editor  
<http://www.textpad.com>
  
- [3] BBEdit for Macintosh  
<http://www.barebones.com>
  
- [4] IBM WebSphere Studio Application Developer  
<http://www-3.ibm.com/software/ad/studioappdev/>
  
- [5] IBM Scholar Program  
<http://www-3.ibm.com/software/info/university/>
  
- [6] Eclipse - open source extensible IDE  
<http://www.eclipse.org>
  
- [7] Cocoa development environment  
<http://developer.apple.com/cocoa/index.html>  
<http://developer.apple.com/techpubs/macosx/Cocoa/JavaTutorial/index.html>
  
- [8] Apple WebObjects framework  
<http://www.apple.com/webobjects>
  
- [9] Designing Integrated Solutions for eBusiness  
<http://www.sitacs.uow.edu.au/ebusiness>
  
- [10] IMS Learning Design Best Practices  
<http://www.imsproject.org/learningdesign>

## ***The Java Infrastructure for Rart, a New Art Form***

by Jan Aminoff , MSEE

### **Abstract**

This paper describes the framework for a new art form, Rart, where a work of art is created as a Java program. A work of Rart, called a universe, is a Java class descended from the abstract class **Universe**. A universe is relatively easy to program since it only needs a subset of Java.

A Rart universe descending from **Universe** can be executed in a rarrunner. They come in several flavors with varying capabilities for the observer to modify the visual impact of the universe through parameters. A universe can be observed over the Internet using one type of rarrunner. A universe may also be observed executing in a rarrunner application with a slicker interface and the ability for the observer to create unique hardcopy of a view frozen in time. The rarrunners also ensure that a universe will remain executable over time and across new and changing platforms.

Since Rart uses a limited subset of Java and since a Rart universe is an understandable and visually appealing concept, we suggest that Rart can be used as a vehicle to teach introductory Java, either as a part of a full length course or as an independent study project.

### **Introduction**

Rart® (Random Art) is a new form of art. A work in the new art form is a computer program generating dynamically animated, constantly changing images. The image-generating program is called a Rart *universe*. One can think of the computer screen as a two-dimensional window to a visually interesting universe. The creator of a Rart universe is a *artist*. The *observer* of a Rart universe can interact with the universe through artist-defined parameters.

Our long-term ambition is to have Rart accepted as a new art form by a worldwide community of artists. This should be possible because of the pervasiveness of the Internet as long as the means to become a artist are simple, inexpensive, and easily accessible. The creation of a Rart universe should also be easy for any programmer including those inexperienced or self-educated. This is one reason why we choose the JICC as a forum to introduce Rart internationally and to have interested Java teachers provide input for further development and dissemination of Rart. If learning Java for a limited purpose is easy and attractive to students, then it should also be attractive to teachers. We hope that by showing the mechanics of Rart, as well as providing actual examples, we can show that Rart indeed may inspire students to dig into the details of an API, that is AWT, the Abstract Windowing Toolkit, and, when they use it, also pick up the details of the new language.

Anyone with a personal computer and a Java enabled browser (for example, Internet Explorer 4.0 or better) can observe a Rart Universe over the Internet, since it may execute in an applet environment. Also, anyone may download for free all software needed for the development of Rart Universes; Minimum requirements are JDK 1.1.8 from Sun Microsystems and the *Rart Development Kit* version 2.0, RDK 2.0, from the Rart web pages. ([www.rart.com](http://www.rart.com)).

A Rart Universe can also, without recompilation, execute in an application environment provided in RDK 2.0. This gives the observer a better user interface as well as the ability to freeze and print a hardcopy of a frozen view of the universe. A compiled universe may be distributed for execution as an application with any Java runtime environment later than, and backward compatible with, JRE 1.3 from Sun Microsystems. The addition of runtime execution features that may include authentication is the basis for a potential distribution mechanism where a artist may be remunerated for his/her copyrighted work. In this paper we discuss the Rart Universe as a Java program, or more precisely, the Java aspects of any universe developed in the RDK. We also provide the essentials of the various execution environments for Rart Universes.

## Rart as an Introduction to Java

This forum has seen many good discussions about Java, its suitability for teaching programming in general and object oriented programming in particular. By way of introduction, I would like to quote from three papers presented at JICC6.

In his paper, Dave Collins, Keele University, "contends that Java introduces too many complexities too early in the learning process". He goes on to describe a learning environment where the first part is based in VBA, Visual Basic for Applications, and Java is introduced as a remedy for shortfalls in the VBA environment [Java Second, The Suitability of Java as a First Programming Language]. David Grey and Rob Miles, University of Hull, discuss the same problem and their approach in developing an interactive web-based course. "By representing real world objects by pre-written classes, . . . it is possible to use these in practical exercises. Nothing is hidden from the students when early programs are described in the text, but the students are told which parts they should concentrate upon. One of the advantages of an interactive course is that students can be given partially completed, working programs upon which they can base their practical work." [Teaching Java Objectively – Reflections on a Web-based Java Course]. Finally, I would like to refer to Alan O'Callaghan, De Montfort University. His paper states that it is "practically impossible to deliver the modern scope of Java within the framework of the traditional degree structure and the 'bottom up', 'syntax first' approach to teaching . . ." O'Callaghan goes on to provide three proposals for computer science. One of these is the "Design Studio Concept" where students collaborate to provide useful and usable solutions under the mentoring of senior teachers. [Redesigning Computer Science].

In order to be productive using Java, you must understand both the application area for which you are to build a system and the tools, the Application Programming Interface, API, allowing you to build useful systems in the application area. To learn one API in detail, you will have to learn how to find the documentation, how to read source code, and, most importantly, find sources where similar applications have been programmed and documented. This may be a daunting task, especially since the state of the art is a moving target where new APIs are added to the pool of Java APIs at a rate far exceeding the capacity for formalizing their adoption, as suggested by Collins.

Rart provides a form and a forum for artistic expression. Rart is inspired by the Internet with Java as the enabler. Students who learn Java for the purpose of Rart, study the Abstract Windowing Toolkit. AWT has been around since JDK 1.0 (1996) and is stable, well established, and well documented. Students also work developing software that can be shown in Java applets. This is because I feel that one of the most important aspects of Java is the fact that it is enabled on the Internet, allowing learning, collaboration, and display of creativity independent of location and time. I see this as an Internet base for a worldwide Design Studio as proposed by O'Callaghan.

I have developed and provided a web-based introduction to Java and Rart since 2000. A recent upgrade of the Rart software, now RDK 2.0, has resulted in a rewrite of the course. The introductory part of the course can be taken on the web with no requirement other than a Java enabled browser and JDK 1.1.8 ([www.rart.com/JavaIntro030115/](http://www.rart.com/JavaIntro030115/)). In my experience, students appreciate being exposed to functioning code and learning that the edit, compile and run cycle actually works. The fast turnaround resulting from a compile cycle measured in seconds also causes students to rapidly assimilate what has for some has been a hurdle, new notations and unfamiliar, unforgiving syntax. The misplacement of a semicolon or the omission of a bracket can be rapidly detected and corrected. Since the student is learning a small subset well, he or she can quickly with the teachers guidance develop a methodology for researching the details of an API by going to the source or the original documentation (javadoc). This is altogether a positive experience for the students and corroborates the observations of David Grey and Rob Miles.

When programming a Rart universe, the student gets immediate visual feedback on what changes have been made. When finished with the course the student has the satisfaction of having developed a work of art. Granted, the pretty applets may not be immediately useful to a potential employer. However, the student will have an important skill needed to become useful and productive, that of mastering the art of discovering what a new API is about. And the student will have the self-confidence that comes from having created something available for the world to see on the Internet.

## Java as Used for Rart

The first Rart implementation, embodied in RDK 1.0, was compatible with JDK 1.0.2, which was released by Sun in 1996 and used as a basis for the first browser implementations, in particular, Netscape Navigator 2.0. We decided, however, to make the second implementation of RDK compatible with Java 1.1, which is more applet friendly in that it allows the consolidation of many class files and resource files in jars. For international deployment, it is important to minimize download time, and today's browsers almost universally use Java 1.1, which supports jar-files. For example, Internet Explorer has used the same Java engine since version 3.5. Accordingly we decided that any Rart Universe should be developed under JDK 1.1.8 which is quite sufficient for most graphics programming and in itself simplifies matters considerably when spreading the Rart paradigm since mature and stable programs and application programming interfaces are well documented and readily available.

A Rart Universe is executed in an environment called *rarrunner*, and the rarrunner takes care of many of the tasks that otherwise would have to be programmed individually. In particular, the rarrunner provides all user interfaces, which actually are limited to mouse activities, and all multithreading. Conceptually, the rartist only has to consider what his/her universe does as the rarrunner invokes its main cycle over and over again. An explanation of the parts of a Rart universe should make this clear.

## The Rart universe

Technically speaking, a Rart universe is defined through the abstract class **Universe**; that is, any executable universe (lowercase u) extends **Universe** (Java class, upper case) by providing the definition of a number of methods declared abstract in **Universe**.

## Listing of Universe

**Universe** is defined as follows (a complete and more commented version is available on the Rart web site):

```
package rartbase;
import java.awt.*;
import java.util.*;
import java.awt.image.*;

public abstract class Universe extends Component{

    // Section 1 Identifying the universe
    // Three simple public abstract methods getName, getDescription, getRartist

    // Section 2 What the universe does
    // public abstract methods init, cycle, manageChange
    // . . .

    // Section 3 uParameters as used in the universe
    // uParameters provide the mechanism to interact with the universe
    public static final int NUMBER_OF_PARAMETERS = 10;
    public uParameter uPs[] = new uParameter[NUMBER_OF_PARAMETERS];
    public uParameter[] getuParameters(){
        return uPs;
    }
    // . . .

    // Section 4 Utility Methods
    // RND, SIGN, and ABS provide a unified way to describe randomness
    // printString is used for standardized output of text
    // setResourceObjects is used to provide the names of text, images or sounds
    // to be used in the universe

    // Section 5 Useful variables related to the execution of a universe
    // int nc, a counter of cyxcles executed, set by the rarrunner
    // boolean doubleBuffer, indicates automatic doublebuffering
    // rarrunner, indicates the class of the current rarrunner
    // int xm, ym, the size of the current window
    // . . .
```

```
// Section 6  Methods related to security.  
// these methods are of limited interest to rartists  
  
}      // End of abstract class Universe
```

The section headings below refer to the section headings of the code for **Universe** as listed above.

### Identifying the universe

The section is provided to recognize that a universe is a work of art. The rartist gives the universe a name, not necessarily that of the Java class, and a short description as well as their preferred name or handle. These strings are presented to the observer in various ways, including an "About" dialog.

### What the universe does

The section provides the methods that define the appearance of the universe.

**init** does all the necessary initiations. It is often done in two parts, **getParam** and **restart**, where **getParam** finds the observer-provided initial values of any defined parameters, either from the applet tag when executing in an applet or for some parameters on the command line for an application. The **restart** method is defined here and will do the initiation using current parameters.

**cycle(g)**, where **g** is a graphics context defined by the rartrunner. **cycle** provides what is actually showing on the screen using methods for line and text drawing and image painting provided by AWT, the Abstract Windowing Toolkit. It is up to the rartist to decide for example if the screen should be completely redrawn each cycle or if the cycle should provide only incremental changes. As can be seen from the examples, much can be done with the simple methods available. The **cycle** method will be programmed using the current values of the screen size provided as **xm**, **ym** (see below).

**manageChange(uP)**, where **uP** is a **uParameter**, is coded to respond changes in the environment or applicable parameters. One **uParameter** is used by the rartrunner to indicate a change in screen size. This particular change is likely to result in a call to the restart method mentioned above. **manageChange** is typically coded with a switch with cases selected from the index to the relevant **uParameter** in the **uPs** array of **parameters**.

### Parameters as used in the universe

The section defines an array of up to nine elements, type **uParameter** used for communication with the rartrunners. A **uParameter** in RDK 2.0 is of one of two kinds, either an integer in a rartist-defined range, or what to the observer appears as a selection of a property from a rartist defined set of properties. The rartist defines a string giving a descriptive name of the **uParameter**, a short description of its function, and a default initial value. These entities appear in a standard dialog for modification of parameters. At any time, a **uParameter** has a current value which is accessed in the universe, for example, through **number.getCur()** if **number** is the name of the **uParameter** class or through **uPs(NUMBER).getCur()**. For parameters of the second kind, the **uParameter** current value is the integer index to an element in the array of selection alternatives given with the definition of the **uParameter**.

Some parameters are predefined. A predefined **uParameter** is **cycletime**. The rartist defines the range of times between subsequent calls to **cycle**, but the name as well as the description is provided in **Universe**. Undefined elements of **uPs** are given the value **null**. The indices have actually been given names both as variables and as strings. **NUMBER** is the index to the **uPs** element which presumably has been defined by the rartist as a **uParameter** to define the number of something relevant for the universe, for example the number of simultaneous lines in the *Lines* universe. The string "**NUMBER**" is used in the applet tag to set the initial value of the **uParameter** with the index **NUMBER** in the **uPs** array.

## Utility Methods

The section defines some simple functions with names taken from the same functions in QuickBasic. We have also provided **printString** as a standard way to output text. Text, which may be multi-line with explicit line breaks, is output in blue, bold sans serif, and in a point size given by the rartist.

This is also where we handle the import of resources, such as files for text, sound ( au format in RDK 2.0), or images ( gif format). In the universe, the rartist provides the names of the files with the specified resources giving a new value to an array defined as **protected String[] reFileNames = null;**. The rartrunner gets the list of filenames through **public String[] getreFileNames(){ return reFileNames;}** and procures the requested resources which are provided to the universe through **public void setResourceObjects( Object[] os){ resourceObjects = os; reFileNames =null;}** with the array **resourceObjects** defined as **protected Object[] resourceObjects = null;**. The reason the task of getting resources can not be handled in the universe itself has to do with security. An applet can only have access to resource files associated with the server from where it was downloaded. The universe is not an applet, but when it is executing in a rartrunner applet the rartrunner has access and the procurement of resources is allowed. It is, of course, up to the universe to deal with the **resourceObjects**, including the casting of the objects to the correct resource. Since the **init** method will set the **refileNames**, the resources should be processed after **init**, but before **cycle** starts repeated execution. This complication is unfortunate and rather counterintuitive, and we believe beginning students should refrain from importing resources all together.

## Useful Variables

The rartrunners make information available to the executing universe. **nc** provides the number of executed cycles, (actually modulo 2000000 to prevent overflow). If the rartist wishes to execute the universe without the benefit of double buffering, he/she may set **doubleBuffer** to false. In some instances it may be useful to know the current rartrunner, for example. if the rartist wants a slightly different behavior for the universe running in an applet or in an application. However, the most important used variables are **xm** and **ym**, which give the size of the current window. While the size of the window is fixed in a browser environment, it may vary when the universe is running in an applet in an applet viewer or when it is running in a rartrunner application. It is necessary for the rartist to take into account the size of the window especially when its size is changed, which fact is communicated to the universe through the **uParameter** with index **VIEWSIZE**.

## Methods related to Security

The section contains methods intended to insure that the executing processor is not brought to its knees by mistake or evil intent. The methods ensure, for example, that only one copy of a universe can be running at any time and that cloning or other duplication is prohibited.

## A Summary of Java as used for a Rart Universe

The definition of a universe should be relatively stable over time to ensure that the work of art created also lasts and remains executable. We repeat here what we said above that we deliberately have selected an early Java version for the universes and removed the need for the programming of the user interface as well as the need for dealing with threads, exception handling, and input-output. A beginning student can for this reason concentrate on syntax and the handling of graphics. My experience is also that it helps that almost all numeric variables are either integers or, only occasionally, long integer.

## The Executing Environments

The function of a rartrunner is to service the universe and effectively isolate the universe from an operating environment that may vary over time and with the introduction of new Java features. The rartrunners can for this reason provide whatever the current Java environment supports and may be modified to use new Java features that would make the life of the universe observer more pleasurable.

The Rart runtime environment in RDK 2.0 is provided in three packages, **rartbase**, **rrlet**, and **rr11**. The package **rartbase** is always needed, the package **rrlet** is needed to execute a universe in an applet, and **rr11**

is needed in addition to execute a universe in an application in a JDK 1.1 environment. We shall briefly describe the functions of these packages.

## Simple Applet

The package **rartbase** contains among others the classes: **RR** and **RRL** as well as **Universe** as we have seen above and the **uParameter** class which provides the **uParameter** functionality.

**RR** is the basic rartrunner with very limited capabilities. It does, however, allow the execution of a universe as an applet. The declaration of **RR** reads:

```
public class RR extends Applet implements Runnable{ // . . . }
```

This rartrunner needs to be an applet in order to execute universes over the Internet. **RR** can, in fact, be used to start a universe, say **MyUniverse**, with the following applet tag:

```
<APPLET CODE="rartbase.RR.class" WIDTH="760" HEIGHT="450">  
<PARAM NAME="UNIVERSE" VALUE="MyUniverse">  
<PARAM NAME="DEBUG_LEVEL" VALUE="2">  
</APPLET>
```

**RR** allows the parameters **WIDTH**, **HEIGHT**, **UNIVERSE**, and **DEBUG\_LEVEL**, where **UNIVERSE** provides the (Java class-) name of the universe and **DEBUG\_LEVEL** is used to turn on and off up to five levels of debugging output during development. What **RR** does not do is deal with parameters (except for any change of window size, which is handled through the **uParameter** with index **VIEWSIZE**). This means that the universe will execute with default values that cannot be modified. **RR** does, however, deal with resources, at least images and text. Sound files are trickier and are not yet handled correctly as of January 2003. **RR** also supports the "About" dialog which presents the rartist provided information about the universe and is activated through a "long" (longer than 500 ms) mouse click.

**RRL**, which stands for RartRunner Light, is intended for the display of universes in a fixed environment such as a web page. **RRL** extends **RR** and adds two methods, **getHTMLuParameters()** and **doubleClick()**.

The first method is used to access values provided in the applet tag. If, for example, the **uParameter** with index **NUMBER** is defined for the universe it can be given a value by the addition of the line **<PARAM NAME="NUMBER" VALUE="22">** in the applet tag suggested above. This gives the **uParameter** the value 22, if 22 is in the range defined for the **uParameter**. If the value is outside the range, it defaults to the maximum or minimum value of the range if the value is greater or smaller, respectively. So for **RRL**, the way to execute the universe with different values for the **parameters**, is to edit the startup html document and restart the universe with **uParameter** values as desired, also of course, invoking **CODE="rartbase.RRL.class"**

However, we have decided to allow one **uParameter** to be modified by the observer. This is done through the **doubleClick** method, but only for the **uParameter** with index **NUMBER**. When the user double clicks the applet, the **uParameter** is modified such that one tenth of the permitted range is added to the current value modulo the upper range. This has the effect of cycling the allowed range of values in ten steps.

## A complete Applet

The package **rrlet** adds the classes that allow full deployment over the Internet. The class **RRlet** is the rartrunner, and **RRlet** extends **RRL**. The package also contains classes to handle the various dialogs. A long click invokes the "Select Action" dialog which allows the observer to select among a number of actions, one of which is "Change Parameter." This leads to another dialog box listing the names of the parameters of the universe. When an observer selects one, a final dialog box appears with the rartist provided description and a means to modify the current value of the **uParameter**. For the first type of **uParameter**, the observer drags a scrollbar, for the second type he/she selects from a list of alternatives. This may all seem somewhat awkward, but it is to be noted that the **RRlet** rartrunner has been programmed in Java 1.0 2. It can thus with minor modifications to **RR** be used with really old browsers.

The **rrlet** package also contains a specialized and customizable rarrunner **rrletDemo**, which is used to create continuously running predefined universes selected at random in the environment of a demonstration. **rrletDemo** extends **rrlet** and adds facilities to interrupt the execution and includes hard coded lists of universes to be executed.

### **A complete Application**

The package **rr11** includes the remaining classes to allow the execution of a universe in an application environment. The rarrunner is called **RR11** since it really required the functionality of Java 1.1 to be operational. It extends **RRlet** and adds a number of fairly large and complex classes.

In order to provide the ability to select from among available universes each universe comes with a short file with the same (Java) name as the universe, but the extension **.unv**. Possible future uses of the **unv** file for security and authentication is beyond the scope of this paper.

**RR11** starts up a banner from which the observer escapes through a long click that invokes a file selection box. In the file selection box, a list of the files with **unv** extensions appears. Once a universe is selected, it starts up using default values for any parameters. It is in the application case not possible to provide non default values on the command line except for **WIDTH**, **HEIGHT**, **UNIVERSE** and **DEBUG\_LEVEL**. The option to select another universe for execution through the same file dialog is available once the rarrunner gets started, and the modification of any parameters is also simple.

The user interface is now a multilevel popup menu activated with a long mouse click. The options lead to the same parameter change dialog as in the **RRlet**, but the convenience is much improved. Two more options are introduced. By selecting "**Freeze**" the observer can freeze and save up to five views of the universe, that is, moments of particular beauty or other interest. The "**Print**" option allows the observer to select from any of the recently saved views of the currently running universe. The selected view can be printed out after the observer has added a title. The printout can be in either landscape or portrait format. In the portrait format, the view is scaled to fit on half the paper making it suitable for mailing as a postcard. In addition to an optional two-line title (or dedication and title) the printout provides a border as well as a timestamp with the moment of selection.

### **The Effects of a Flexible Execution Environment**

We have seen how the rarrunners isolate the universe from its executing environment. It is of no concern to the rartist how or where the universe is executed. A further and in the long run more important benefit is that the rarrunners can evolve with time to take advantage of new Java features as well as new or evolving operating systems. This provides the assurance that any universe will remain executable over time as well as across any present or future platforms with no further effort on part of the rartist. A Rart universe will exist as long as computers are used on Earth.

---

*Jan Aminoff received a Master of Science in Electrical Engineering from Chalmers University in Gothenburg Sweden and another MSEE from the Polytechnic Institute of Brooklyn, New York, in 1972. After a career in telecommunications, he has devoted himself to various aspects of Java since 1996.. He lives in the United States and can be reached through [j.aminoff@computer.org](mailto:j.aminoff@computer.org) or indirectly through the web page <http://www.rart.com/>.*

# Teaching Java programming to media students with a liberal arts background

Jens Bennedsen  
Department of Computer Science,  
University of Aarhus, Denmark  
jbb@daimi.au.dk

## Abstract

During the last years we have developed a curriculum in Multimedia for students with a liberal arts background.

In this curriculum we have had several different programming courses. These programming courses have had different problems – especially with the formalism in the courses.

We thought we knew what the problem was: *Lack of motivation*. The students considered here consist of students to whom programming is not a primary interest and many are prejudicial against programming.

We were mistaken, it was not a motivation problem. We designed a questionnaire to find the students' attitude towards programming, and it is much more positive, but they expect that it is very difficult.

We present the results of the questionnaire and interpretation of the different parts of the questionnaire leading to a new curriculum for the programming course and the criteria's for its design given the attitude we found the students had to programming.

## 1. The curriculum

The students discussed in this article are students taking a bachelors degree in multimedia. They have a liberal arts background and they want to specialize in multimedia. For them we have designed a special third year of their bachelor degree.

This year consists of courses from humanities and computer science and one of the computer science courses is a programming course.

The goal for this year is to give the students enough insight in the multimedia technology and design processes so that they can create professional multimedia products.

Why do these students need programming? Apart from programming being one of the key technologies the ability to be programmed is also the defining characteristic of the media – this is what differ multimedia products from traditional media products (see [1]). If we removed programming from the curriculum we allowed the students to study multimedia products without studying the defining characteristic.

## 2. The previous courses

In the previous years these students have been enrolled first on a traditional CS1 course with our “traditional” computer science students. This was a disaster! After only a few weeks the students were very demotivated and found that they could not master programming at all.

We interviewed them and they said that their main problem was math. The traditional computer science course used some math, and there students have not got a particular good math background, so there was reason to complain.

Our own feeling was that it was not only a math problem, but more a clash between the liberal arts background and science. They have been used to discussing with people where it is the social competences that plays the most important role. This new counterpart – the compiler and this strange language that they have to speak – have none of the traditional competences, like understanding something without being totally precise, they are used to.

In order to see if we were right about our assumption, we discussed the following quotation from [3] with them:

*“First one must perform perfectly. The computer resembles the magic of legend in this respect, too. If one character, one pause, of incantation is not strictly in proper form, the magic doesn't work. Human beings are not accustomed to being perfect, and few areas of*

*human activities demand it. Adjusting to the requirement for perfection is, I think, the most difficult part of learning to program”*

They all agreed that this was a very precise description of their experience with programming

### 3. The attitude to programming

Initially we thought that the students dislike programming and just followed the course because the curriculum told them so. This was just a feeling, so we need to be sure about the feeling. This was done by a questionnaire to the students before they started the course and before the design of the course was final.

The questionnaire was in two parts.

1. Free text questions about the qualifications the students had on before hand, why they had chosen this curriculum, what they think programming is and their attitude towards math.
2. Six phrases about their attitude towards programming. The students should then indicate how much they agreed with the phrase. The phrases were:
  1. Programming is only for nerds
  2. You have programmers to do the programming
  3. Programming is difficult
  4. I prefer to work with graphics and sound instead of programming
  5. Programming is a strange and “cold” activity as opposed to design and aesthetics
  6. I do not need to learn to program in order to create a media product

The questionnaires main focus point was to see how the students initially felt about programming and there motivation for learning to program.

We choose a questionnaire for the following reasons: 1) We wanted to have answers from as many students as possible. 2) We wanted to be able to see if the programming course would change their attitude towards programming. We wanted to know their attitude since we are convinced that initial resistance against programming make the job of teaching programming very hard (especially if you are not aware of the resistance).

We got 85% answers. There are a total of 35 students participating in the questionnaire.

#### 3.1 Free text part

Our expectation was that the students did not like programming, and when they were asked to describe it they would choose negative words in their description. This was not the case – they choose either neutral or positive words.

We thought that most of the students were scared of math and typically have had very bad experience with that topic. This was not true, about 50% of the students answered the question with a positive attitude towards math. But some of these answers also indicated that they find math difficult but still useful.

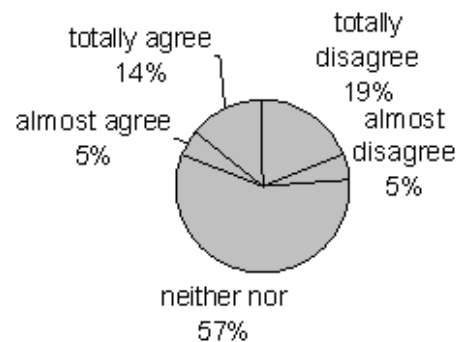
The conclusion is therefore: The students are not negative towards math or programming but they do not have a very good math background.

#### 3.2 Attitude towards programming

This part of the questionnaire should elaborate on the students’ attitude towards programming.

##### 3.2.1 Programming is only for nerds

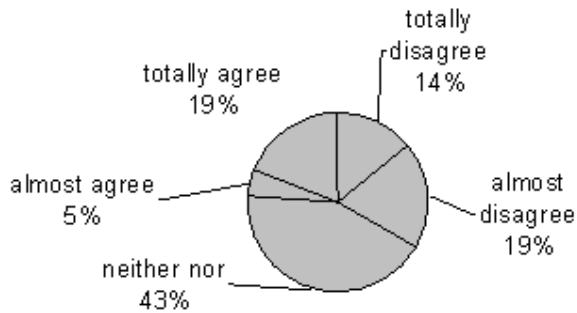
This phrase was introduced so that we could see how these students felt about programming – is it an activity that they will be proud of or is it something that they would rather not do. Nerd is regarded as a negative word.



As you can see from the pie chart there are only 19% of the students that either totally agree or almost agree with this phrase. 57% of the students answered neither nor – possibly reflecting that they do not know what programming is, but do not dislike it. The conclusion is that the students do not dislike programming beforehand.

### 3.2.2 You have programmers to do the programming

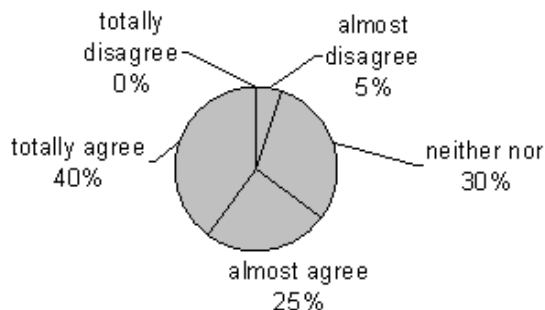
The reason for this phrase was to see if the students could see themselves as programmers. It was also introduced to see if the students think they need to be able to program or they can find someone else to do the job.



As you can see from the pie chart the answer for this question follows the answers to the previous question. Again the conclusion is that the students want to learn to program.

### 3.2.3 Programming is difficult

This question was part of the questionnaire to see how hard the students think it is to learn to program.

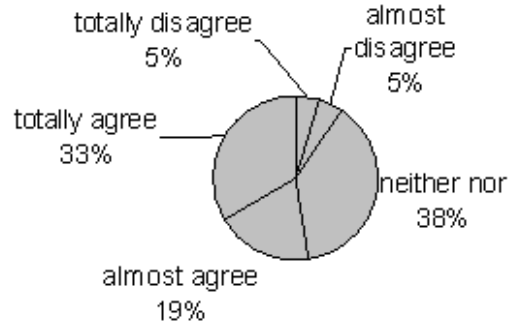


Almost all of the students think that programming is difficult! Why do the students think this it is difficult, when they have not tried it (Almost none had programming qualifications on before hand)? Rumours from the older students!

The conclusion is that we need to move slowly in the beginning of the course so that this difficulty of programming not will become a self fulfilling prophecy – the Rosental effect [8].

### 3.2.4 I prefer to work with graphics and sound instead of programming

This question was part of the questionnaire to see how hard it will be for the programming course to “compete” with the two other courses in the semester (they are about multimedia design and multimedia production)



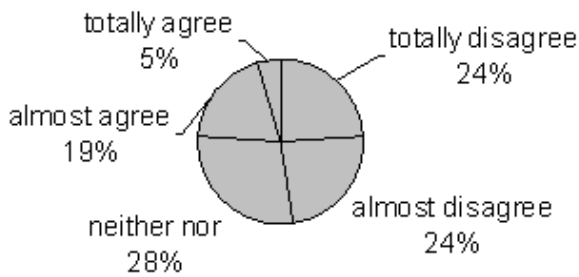
Over 50% of the students agree and only 10% disagree! The reason for this is the liberal arts competences the students think they will use in a multimedia curriculum, namely the design of the interface to a multimedia product.

The conclusion here is twofold:

- 1: We need to make the programming appealing and use graphics, sound and other multimedia means in our programming. This is also true for traditional programming curriculums as discussed in [7].
- 2: Design the programming course in combination with the other courses so that the students can see the use of programming when designing multimedia products.

### 3.2.5 Programming is a strange and “cold” activity as opposed to design and aesthetics

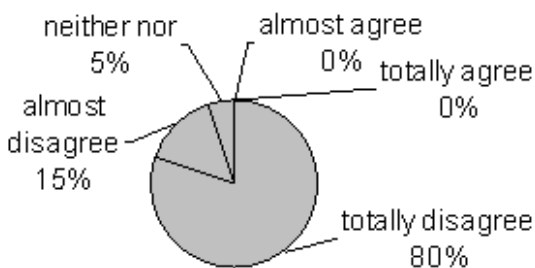
This question captures if the students find programming repulsive, an activity that they would rather not do. If they do there is a big task in motivating the students to use enough time in the programming course.



About half of the students find programming as appealing as design and aesthetics, and only about a quarter of the students find it a cold activity.

### 3.2.6 I do not need to learn to program in order to create a media products

If the students can not see a reason for the programming when they create multimedia products it will be almost impossible to motivate them to learn to program.



Fortunately none of the students think that they do not need to learn to program – they can all see the need for learning to program.

### 3.3 Conclusion

All in all the results show that the students want to learn to program, they think it is difficult, but they also expect they must master it in order to create the media products that is the goal of their education.

As the questionnaire show, many students have an initial interest in programming and find it to be a necessary tool for them to master. This was an interesting thing to observe, because it means that we do not have to convince the students about the necessity of programming. This is a good starting point for the programming course.

## 4. The design of the course

With the results of the questionnaire in mind we designed a programming course to these specific students.

First of all we need to decide what the content of the programming course must be and thereafter the pedagogical design of the course.

### 4.1 The content

The structure of the course is the following:

- 15 weeks of lectures followed by an exam
- Each week there is 3 hours of lectures and 4 hours of lab-exercises
- they take the course together with a course in multimedia design and one in multimedia production

The goal with the curriculum is that the students can create quality multimedia products. The quality is both external and internal External so that the can create user friendly, appealing products. Internal so that the can create maintainable products. In the programming course we are mainly concerned with the internal qualities of the product, so we need to use a programming language that supports encapsulation, abstraction and all the other traditional virtues of program design. On top of that we need a programming language that is popular, has support for multimedia (sound, pictures etc) and can be used for creating multimedia products.

The answer to this is to divide the course in two parts. The first part (about 2/3 of the entire course) uses Java. The second part uses Lingo<sup>1</sup>.

The java part focuses on the basic virtues of program design – process, encapsulation and the like. This gives the students a good understanding of what good internal program quality is.

This focus on internal quality is also used in Lingo - a programming language with much less support for encapsulation and the like.

<sup>1</sup> The programming language of Director, a widely used multimedia creation product from Macromedia

## 4.2 Pedagogical design.

From the experiences with the previous course and from the questionnaire we have the following design criteria for our programming course:

1. Start slow
2. Use graphics, sound and other multimedia means.
3. Use a spiral approach [2]
4. Use “Fill in the blanks” [4,2]
5. The three courses in the semester must be seen as integrated by the students

Each of these criteria will be described in more detail in the following – what it is and why it is needed, based on the evaluation of our students.

### Slow start

Since the students think that programming is hard, we need to start slowly, so that the students get a good experience with programming (see §3.2.3). We do this in (at least) two ways:

1. Just taking the time needed for the students to get the idea of programming.
2. Teaching how to convert a UML model of the system into a working Java program in a very systematic way.

Especially the second part gives these students a path to follow when they create programs – create a UML model of the program and then convert it into a Java program. It is often easier for these students to make a model of the system in UML and then convert it to a program than directly programming it.

The second part of the course (where the programming language Lingo is used) this systematic way of converting an UML model to a Lingo “program” is followed as well. It is fairly easy for the students to learn the basic conversions rules for Lingo. When they have seen them, they can create the basic parts of their Lingo application with good internal quality. Hopefully this also shows the students that the tools and processes they have learned in Java help them when creating multimedia applications.

### Use graphics, sound and other multimedia means

Over 50 % of the students preferred to work with graphics and sound than to program (see §3.2.4). If we can use these means as part of the programming course, it will be easier to motivate the students.

We have created many examples and exercises that use sound and graphics.

When we for example teach iteration, this is done by using a picture abstraction. First of all it is possible to give many examples of iteration by different picture manipulations. This is motivation for the students. We have designed the picture abstraction so the only way you can iterate through it is by using an iterator. At each pixel of the picture you can get an iterator that iterates through the neighbours – so there is iteration within iteration.

This use of the iterator design pattern [6] both frees the students from the traditional exceptions with borders of the picture and shows them a good program design.

Other examples use games as the motivating factor. Again, this choice is motivated by the answers to the questionnaire.

### A spiral approach

Almost all of the students expect programming to be hard! Some of the problems are related to what is expressed in the quotation in the section 2, namely the extreme preciseness and details that is needed in order to program.

So we need to hide as many details as possible in the beginning, just focusing on what is absolutely needed to do the job. This design is also expressed in the pedagogical design patterns “Early bird” and “Spiral” [2]. These are more common design principles, but it is especially true when the students are not used to the accuracy in the communication with the programming language or other more formal languages.

We have therefore designed the course in a spiral way. First we focus on the basics: Classes and objects with getters and setters. Then one container class and one way of iteration through that container (using an iterator) and then slowly add more and more details. But for each detail it is considered whether or not it is needed by the students to fulfil their task – no details just for the sake of details!

### “Fill in the blanks”

When students are presented problems in the beginning of the course, we need to make sure that the problems both have relevance and are at a level they can master. This is mainly because of the answers to the question about the difficulty of programming.

One way we address this problem is the structure of the exercises for the students – use “Fill in the blanks”.

A “Fill in the blanks” exercise is characterized by the following:

- The starting point is a (working) application
- The exercise is described in small steps
- parts of the application is left out as an exercise for the students
- After each step it is possible for the student to check if their solution is correct
- The last part of the exercise is open ended

An example of the concept is described in [5]. For more information see [4].

### **The three courses in the semester must be seen as integrated by the students**

From talking to the older students we know, that one of the very positive things for them is to experience synergy among the courses on the semester.

When we ask them for the three best things they can say about the semester, almost all of them answer the integration between the courses.

With this integration the “relevance problem” (the students do not think they need programming) is not very evident. The programming course is seen as a necessary qualification when one must make a multimedia product.

## **5. Concluding Remarks**

In the beginning we thought that we knew what the problems of the course was, but after a questionnaire we could see, that the problems with the programming course are not just what we thought. The use of gathering information beforehand by interviewing the students gives many insights so that the actual course can be thought with much less frustration – both for the lecturer and the students.

At the end of the course only two students out of 30 had left the course and at the exam only 2 students did not pass. The average mark of the students was about 9<sup>2</sup>. This is very good compared to how the students were doing in the traditional programming course. Of course it is not the same students, but their backgrounds were the same, and there are a fairly large

number of them, so we find it reasonable to conclude that the design of our multimedia programming course is a success.

Another area where we think this course will be relevant is in the growing field of training – many people with a non-science background need to get a programming education. We believe that these people have a similar background as the liberal arts students and therefore also will benefit from the course.

## **6. Acknowledgements**

I would like to thank my colleague Michael Caspersen for many motivation discussions about design of introductory programming courses.

## **7. References**

- [1] Andersen, Peter Bøgh (forthcoming) *Acting Machines*. To appear in Gunnar Liestøl et al. (eds): *Innovations – Media, Methods and Theories*. Cambr., Mass., MIT press
- [2] Bergin, Joe: *Fourteen Pedagogical patterns*, [csis.pacs.edu/~bergin/PedPat1.3.html](http://csis.pacs.edu/~bergin/PedPat1.3.html)
- [3] Brooks, Frederic P., *The Mythical Man-Month: Essays on Software Engineering*, Addison-Wesley Pub Co
- [4] Caspersen, Michael (1996): ”*Programmering med klodser*” DMLF bladet (in danish)
- [5] Caspersen, Michael et al: (2000) *Here, there and everywhere – on the recurring use of turtle graphics in CSI*, Proceedings of the fourth Australasian Computing Education Conference, Melbourne, Australia
- [6] Gamma, Eric et al (1995), *Design Patterns - elements of reusable object-oriented software* Addison-Wesley Pub Co
- [7] Guzdial, Mark et al (2002): *Teaching the Nintendo generation to Program*, Communications of the ACM, April 2002
- [8] Rosental, R. and Jacobson, L. (1968) *Pygmalion in the Classroom: Teacher Expectations and Pupil's intellectual Development*, New York: Holt, Rinehard and Winston,

---

<sup>2</sup> In Denmark marks are from 0 to 13, 13 being outstanding

# **eXtreme Programming (XP) with Java and Jython in the Classroom**

David Dench

School of Computing and Engineering, Huddersfield University

Queensgate, Huddersfield. HD1 3DH

( [d.j.dench@hud.ac.uk](mailto:d.j.dench@hud.ac.uk) )

## **Abstract**

Agile Software Development has a growing profile in the Software Industry. Arguably, the flagship method is Extreme Programming ( XP ). Experiences of using XP in an educational context are few. This paper details a continuing attempt at introducing XP and its practices into a Software Development Module. It offers some guidance on the inherent difficulties in introducing a clash of cultures.

## **Keywords**

Agile Methods, Extreme Programming, Java, Jython, teaching XP

## **1. Introduction**

I was first exposed to programming as an undergraduate in 1973, and have been teaching at Huddersfield since 1986. I have "taught" programming and related software development issues for many years, to students on various courses and at all levels. Teaching introductory programming is notoriously difficult and I am still struggling to find a truly effective way. Teaching Advanced level modules to more practiced students is far easier by comparison. What is obvious is that there is no "best" way for all students and that we lose too many students too early. Lack of support, difficulty and sometimes boredom kill off any enthusiasm long before the students have a chance to attain any level of ability. XP offers some hope in offering support and maintaining enthusiasm, whilst the students attain appropriate levels of self discipline and technical ability.

## **2. extreme Programming( XP )**

Extreme Programming ( XP ) [1] is one of a number of Agile Software Development methods [2] . XP espouses the values of simplicity, communication, feedback and courage and is structured around (originally) twelve key practices. These practices are usually listed as Planning Game, Small Releases, Metaphor, Simple Design, Testing, Continuous Integration, Pair Programming, Collective Ownership, Refactoring, 40-hour week, On-site Customer and Coding Standards. The practices are detailed further in many sources, in particular, Beck [5]. XP is often characterized by its detractors as no-design, hacking. This could not further from the truth. It is a high-discipline, social development method focussing on quality code sufficient for the business needs. Kent Beck, one of the XP pioneers, is a charismatic speaker and his keynote speech at the Object Technology conference at Oxford in 1999 filled me with enthusiasm. The principles and practices laid out by XP seemed to hold out hope for keeping my students fired up, whilst instilling solid software development skills, in particular testing skills. I immersed myself in XP. The XP community is extremely dynamic and enthusiastic and information is abundant ( [1], [3], [4] ).

### **3. XP in the classroom**

There are a number of educators in the UK experimenting with XP in the classroom for example Southampton and Lancaster Universities, but I am not aware of any published reports on using XP as a teaching vehicle in the UK. There are a number of educators worldwide currently introducing XP into the curriculum. Published results are sparse. There is one notable and comprehensive recent report by Ivan Tomek at Acadia University [6]. I was aware that this experiment was proceeding concurrently to my own, but was unaware of the details. Having subsequently read his report and conclusions, I can only say "me too". He details experiences that are remarkably consistent with my own.

### **4. Module Context**

I was convinced that XP held out promise for teaching programming at all levels. There was only one problem, I did not have any appropriate teaching vehicle at that time to try it out. Most of our students have an introductory first year programming grounding in Java, some then study C++ in the second year. Third year is traditionally an Industrial Placement year, although sadly, full time degrees are becoming more common. Our final year modules are usually open to students from any Pathway, and as such we cannot expect every student to have any substantial Java experience. I have a number of years experience teaching using C, C++ and Python as vehicle languages, but had not lead a Java based module. I was aware that by using Java exclusively I would be putting some students at a disadvantage. Getting to grips with a new language as well as new disciplines did not make for a level playing field in assessment terms.

### **5. The first attempt**

I was asked to take over an Advanced module called "Advanced Programming" at short notice just prior to the start of Session in 2000. The existing Module Specification was amenable to shoe-horning in some XP material.

Unfortunately the base disciplines of XP and the constraints of the Module Specification as it stood, seemed mutually inconsistent. I only had twelve two hour sessions. As a core module ( for one Pathway ), I was mandated to have an end of session exam, so I would have to squeeze in a second shorter practical assignment. I had a small number of students (10) on the course from a variety of Pathways and mixed programming backgrounds, but I also had some of our brightest students on the Module.

I was faced with more questions than answers.

Is it possible to teach XP in the classroom ? There were no educational experiences to draw upon. How do you keep interest/ structure it between weekly sessions?

Assignment pressure from other modules can cause havoc with attendance. How do you get across sufficient techniques and knowledge in 12 x2 hourly sessions such that the students are fired with enthusiasm/capable to tackle an assignment mid-Semester and sit a 2 hour exam at the end ? The assignment would of course have to be individually assessed. Problems with group assessment and plagiarism are worryingly common. What techniques can be used to coach weaker students ? I had no teaching support. Did I have problems of sufficient but not overwhelming complexity to enable a true flavour of XP, or would it degenerate to a selection of XP practices ?

As it was such a tight schedule, courage ( or the lack of it ) got the better of me and I fell back on a traditional delivery, delineating the base knowledge, concentrating on some core practices such as Planning Game, Unit Testing and Refactoring ( Using

Java and JUnit ). Practical sessions allowed experimentation with Pair Programming and the Planning Game, but the single assignment was still a short span individual affair. This was a frustrating experience.

Pedagogical Patterns must surely have had a role in developing a suitable module based around XP, but I simply could not see a way around the Module constraints.

The structure just did not seem conducive to an intense, cooperative discipline.

The usual policy on Computer Laboratories allowed student access if there was a free seat. I enforced a ban, and made the room for exclusive XP use. This did allow for a more informal session and allowed me to experiment with toys and games as a way to make the sessions more "fun", but it was only for an hour per week. The University has a ban on smoking eating and drinking in teaching rooms, so I was unable to experiment with making food available at the sessions. I tried to make the sessions as conducive as possible. I set up a set of wireless headphones with "now playing" music that the students could bring in. I encouraged the students to bring interesting puzzles. The traditional wood block puzzles and string-on-a-wire puzzles have good mileage. I had a positive response to the moves, attendance was good and the sessions were lively. However it was overkill as the students did not really have the time to take advantage of the facilities as well as getting a decent amount of work done.

The only thing that I could not affect was the room layout. We had to use one of our standard ( Unix ) laboratories and typically had fixed banks of terminals, not entirely conducive to pair programming. Pair programming was encouraged but was not forced.

For the assignment, I used an old problem that the students had encountered in a previous module, a fairly standard Football League system. I did this for a number of reasons. The fact that the students were familiar with it, led to a simple "naive" metaphor. I acted as customer for the planning game and generated the stories which the students then estimated. I intended to work on a 1 week = 1 story point iteration and split the stories to enable this to happen. The students paired during the sessions and attempted Test-Driven Development (TDD) [7] during the session, but pairing was not mandatory. I encouraged the students to attempt a more complete XP experience, but allowed students to opt-out from the planning game and pair-programming. TDD and the use of Java, JUnit ( the Java Unit Testing framework ) and CVS (Concurrent Versioning System) for version control was however mandatory. Each student had their own CVS project to work within, and this was used as a possible tool to stop plagiarism. In CVS, all code amendments are logged and maintained.

The steps taken for plagiarism control and the regulatory requirement for individual assessment are obviously at odds with the XP practices of collective code ownership and pair programming.

Sadly a number of students opted out of the XP experiment.

As the module progressed, I felt that the spirit of XP just was not coming across.

It inevitably became an example in anti-patterns namely " Blind leading the Blind" and "See one, do one, teach one" .

I attended OT2000 whilst the experiment was progressing and I contacted a number of XP trainers and practitioners for help and advice. I received a number of helpful tips and examples. One tip I did try, was enforcing one person using the keyboard, one using the mouse, to force communication in the pairing sessions. The consensus was that I really needed a longer more realistic and properly cooperative XP project. Otherwise, what I was doing was deemed reasonable.

## **6. First experiences**

All the students passed the module. However, many of the best students failed to achieve as high a mark in the practical component as I expected. Of those students that attempted the XP experience, there was a general enthusiasm for the topic, but application was not entirely rigorous. Refactoring [8] tended to be a poor relation. As XP takes a cooperative view, the failures were obviously my fault (even though my name is not Chet [9]) and realistically due to lack of adequate coaching. TDD could reasonably have been expected to be more rigorously applied.

## **7. The Sabbatical**

To raise my own personal experience level, I applied for, and got a single Semester Sabbatical. The Sabbatical had two main objectives, to develop a teaching tool to enable me to move my graphics teaching to Java3D and to practice XP.

The teaching tool that I was to develop (Java3DBuilder) enables students to build Java3D scene graph models (using a fairly simple GUI), to view the scene graph and to generate code for that scene graph. Some of the students opting for the target Graphics Module were not strong programmers, so needed substantial code generation support. This tool enabled the students to experiment with scene graphs and generate code that they could cut and paste into their own applications.

Java3DBuilder concentrates on a fairly simple subset of Java3D,

I generated the tool using a controlled XP-for-one process (I could not find anyone who could pair on a regular basis). XP for one requires substantially more discipline than is usual. I have a lot of sympathy for the self-control required to stick absolutely to the principles of XP. Pairing is a realistic necessity to avoid falling back into bad habits.

The tool was developed in Jython[10] using Swing for the GUI and unittest, the python equivalent of Junit, for Unit Testing. Version control was managed by CVS. The source code, the set of unit tests and the CVS repository was part of the deliverables to be used for the second run of my Advanced Programming module.

## **8. The Second Attempt**

Prior to attempting a second run of the module, I was able to renegotiate the Module Specification. I made the material more obviously based around XP, and most importantly, I managed to remove the examination and assess the module based on a single Group project. I still had to assess the students individually, which became the single hardest issue to resolve.

I maintained the first half of the module as a fairly traditional module, based on XP practice lectures and practicals to develop the necessary skills. The practical sessions enforced an XP approach and coaching tricks were used to make the group more cohesive. Not all the students knew each other, despite being in the final year, as they came from different Pathways. A "nerf" day, where I brought in my children's nerf guns proved a big hit. (Nerf toys are weapons that fire soft foam projectiles). I attempted as many tricks as I could to keep enthusiasm to attend. Music and puzzles were again used, but food and drink is still unfortunately banned.

The second half of the module was devoted to a substantial project that formed the sole assessment of the module. The XP project, run along strict XP rules, was the biggest change. I made attendance at every session compulsory and was used in the assessment of the module. I got just about 100% attendance from all the sixteen students on the module. The students seemed generally keen on attending.

I had one big team, rather than attempting to split it up into two smaller teams, this was for simplicity, but had the advantage that there was a bigger pool of potential pairs outside the class hours. I am not sure if this would scale to more students. I decided to model the project around stories to enhance the Java3DBuilder tool with new functionality. The advantages to this were that they had a good start ( "Fixer-Upper" pedagogical pattern ), a set of unit tests to criticize and enhance, source code that had a full development history on-line ( via cvs2html [11] ) and a client on tap (me) that could help. The students could also see an obvious use for the improvements.

I ran the project with a three week iteration, hoping to get two full iterations completed by the end of the Semester.

The use of Jython was encouraged, but not mandatory. I attempted to level the playing field by giving them all an extra hurdle ( a new language to learn ). The fact that the Java3Dbuilder tool was developed in Jython and more importantly that code layout is part of the Jython/Python language syntax, encouraged the Coding Standards practice. Jython also enabled Java interworking. I hoped that this would also encourage more group cohesion and pairing. Some of the students were taking the Graphics module and were familiar with the concepts and the original version of the Java3DBuilder tool. This was a bonus. This raised the skill levels within the team and those students became primary pairing targets to disseminate their knowledge. This was encouraged. The naive metaphor was extensively used , but an early file system metaphor was adopted to explain the scene graph (but later dropped).

Only one story was developed using Java as opposed to Jython calling Java. This was the addition of a scene graph tree display which was implemented using a JTree. The student who played the major part in it's development ( a bright student) had some prior knowledge, that he brought to bear and it became a case of TSTTCPW ( The Simplest Thing That Could Possibly Work ) .

I kept one "Big Chart" at every session. This was the pairing matrix, that showed who had paired, how many times and in what capacity ( driver/navigator ). This was completed by the students at the start of each session, and was used in the assessment of their individual contributions. The chart was used to pick pairing partners for the sessions.

The stories (on cards) and a flip chart were available at each session and I held them in my office should anyone need them outside of class hours.

During the second run I organised an informal BOF at OT2002 to discuss XP with fellow educators and interested practitioners. Most of the problems I were encountering, seemed common to all.

Grading was mostly based on an individual submission of a report. An evaluation of the process was marked together with contributions based on the quantity of work ( e.g. number of pairings, number of test cases ), quality of work ( e.g. refactorings attempted ) and attendance.

## **9. Further Experiences**

I had 100% attendance except for one or two cases of illness. The students were very keen to attend and made great efforts to excuse their absences when they occurred.

I am not sure how I would have referred a student should they have had an extended absence or failed to cope.

The outcome was a reasonable success. All the students passed the module. Most of the stories were implemented on schedule. ( But I still had to do some fixing prior to using Version 2 of the tool , their unit tests were not exhaustive ! )

All the twelve XP practices were addressed to some extent.

The Planning sessions were a little protracted possibly due to the nature and complexity of the Java3DBuilder software. The trade off with using an existing system and tests is that the knowledge of the system requires time to assimilate and migrate around the group.

This did happen slowly, but maybe I could have accelerated that knowledge earlier by leading quick design sessions.

It seemed to have been a positive experience for most of the students, but I heard tell of some slippage from the self-discipline required by XP. It was not always TDD when away from the class.

Pairing outside the class proved problematical, finding a pair and finding the person who signed up for a Story/tasks was not easy and usually required a broadcast e-mail and its inherent delay. We had one particular problem with a student who was a part-time student and went out of contact for a period. Students were reticent about applying the collective code ownership principle.

Students failed to check in their code to CVS on a regular basis. Code ownership seems in-grained and hard to shift. We had one notable case of a student checking in untested code. This led to a week of frustration in trying to reconcile the CVS repository. XP practitioners at a local XP User Group testified to this not being an uncommon trait in new team members.

We also had to institute a CVS check-in flag. This was a physical flag that a pair grabbed and placed on their terminals to indicate they were checking in. This cured a problem with frustration during the class sessions, where many pairs were attempting concurrent integrations. It did not seem to be a problem outside of the class.

Analysis of the tests showed some lack of clarity in test design and bigger problems with lack of automated GUI and Acceptance tests. This could be put down to my own failings as a coach, and failure to instigate a framework, sufficient examples or guidance as to how to automate such tests. Acceptance tests degenerated into what I believe is called "IdiotChecksOutput". The students were made aware of "mocking" and the availability of the Java Robot class, but this was not encouraged sufficiently. More substantial tools for automated GUI testing exist, but were either incompatible with our environment or expensive. However, these areas do seem to be problematical in the XP community in general, so further development in this area would be appropriate. There certainly seems to be scope for a tool suitable for educational purposes.

Pairing was almost universally greeted with enthusiasm after learning the communication skills that are inherent in being a good pair. The sessions were always lively affairs.

## **10. The next attempt?**

Whilst the experience has proven interesting, and I have plans to run the module in a similar vein, current development has been shelved due to a Pathway re-organisation within our Department. The Advanced Programming Module has been re-written as an "Agile Methods" module but has been made optional. Due to the cross Pathway nature of the module, this means that it will not run in the current Academic Year.

I am also negotiating a change to make the amended module a full year, twenty credit module. This will give more time to acquire base skills and have a longer project.

The skills that the students pick up make them much better developers and I have a possible plan to use the graduates of this module as student mentors for more junior students on our courses, this may be the most beneficial outcome of this module.

An earlier TDD based module also seems feasible, which would fill a testing hole in our current modules. An introductory programming module based on a TDD approach holds out great promise.

## 11. Conclusions

The single biggest problem with using XP in the classroom is the lack of coherent time and inconsistency with development methods expected in other modules. Environmentally XP seems at odds to the traditional classroom mentality. Whereas pair programming is mandatory in XP, it is seen as evidence of Collusion in other modules. Terminal rooms are usually quiet places in supervised sessions, but are extremely lively during XP sessions. The students find it hard to swap modes between classes and inevitably fall into bad habits. I would like to be able to structure the module such that there were more hours per week but timetabling this would be a nightmare.. The students consistently complained that they were just “getting in the groove” when they had to stop and go to another class. TDD has a trick of leaving a failing test to enable the developer to pick up the thread on their return. This might prove effective.

It has been suggested that the XP project could be conducted outside of the teaching weeks [6]. I am not sure that this is practical in our teaching environment, but the thought of large tracts of time devoted entirely to XP is seductive.

Pair programming may also provide a tool to alleviating our wastage rates. Weaker students already seek out stronger students, why not make this a feature and subvert the collusion.

There is also a general lack of coaches available. This can only addressed by teaching support. Financing this is a problem. Coaching requires bodies and time.

Having been through two iterations, it would appear that the XP experience is a positive one. The self-discipline required and testing/refactoring knowledge acquired are worthy in their own right. The cooperation and communication skills that XP fosters are also major factors in making the students more employable. The biggest obstacles to overcome are the lack of coherent time and initial technical support. This experiment continues.

## 12 References

1. Extreme programming website, <http://www.xprogramming.com>
2. The Agile Alliance website, <http://agilemanifesto.org>
3. The XP maillist, <http://groups.yahoo.com/group/extremeprogramming/>
4. Object Mentor, <http://www.objectmentor.com>
5. Beck K : Extreme Programming Explained, Addison Wesley, 2001
6. Tomek I : What I learned Teaching XP, <http://www.whysmalltalk.com/articles/teachingxp.htm>
7. Beck K : Test-Driven Development, Addison Wesley 2002
8. Fowler M : Refactoring, Addison Wesley 2000
9. Jeffries R., Anderson A :“It’s Chet’s Fault” , chapter in Extreme Programming Installed, Addison Wesley 2001
10. Jython home page , <http://www.jython.org>
11. cvs2html tool homepage , <http://cvs.sslug.dk/cvs2html>



## **A Distance Learning OOPs Course**

**T. F. Higginbotham, Ph.D.**  
**Southeastern Louisiana University**

### **Abstract**

During the summer of 2002, I offered a course in OOPs programming, which was 100% via the Internet. No classes met, examinations, problems, programs, discussions, etc., were all online. The delivery system was BlackBoard, which is used by most of the universities in Louisiana.

The problem of getting students to read the text was approached via the use of short quizzes, which seemed to work. Short programs, simple in nature, were used to emphasize OOPs concepts. A mid term and a final examination were given.

Submissions, in spite of these students being experienced C++ programmers, was a problem for some of them. Directions were not followed until I just refused to accept anything not in proper form.

Cheating (copying) programs was not a problem. Class size and pride both may have had something to do with this. Cheating on quizzes did not seem to be a problem, although I have no doubt some did take place. Cheating on the midterm and the final did not occur, based on item analysis

There were some rough spots such as the students did not like the text., and getting started was a problem because of the shortness of the 8-week summer semester.

The amount of work required of the instructor is greater in a distance learning course than in a normal course.

T. F. Higginbotham, Ph. D.  
Professor of Computer Science  
Box 10763  
Southeastern Louisiana University  
Hammond, Louisiana 70402

Higginbotham@selu.edu  
work +1 (504) - 549 - 2055  
fax +1 (504) - 549 - 3396  
home +1 (504) - 345 - 2697

The purpose of Computer Science 355, a third-year course (Our degrees are four-year. Numbers express the level; that is 100 numbers are first year, 200, second-years, etc.) is to teach object-oriented programming techniques thoroughly. The use of C++ does not lend itself to object orientation as well as Java because C++ can be programmed completely from a procedural standpoint. Another reason to use Java is the job market. If you ask the majority of our students why they are here, you will be told it is because employment prospects are much higher. They are quite correct.

The selection of a text for an online course is most important as the students do not have the classroom to clarify concepts and ask questions. Everything must be done in writing. The official text was *Beginning Java 2*, by Andrea Steelman, Mike Murach and Associates, 2002. The innovation of the text is to have discussion on one page and the facing page to have the code. Both the students (I always ask students' advice on text selection.) and I thought this was a good idea. Unfortunately, by the end of the course, the students had decided that this made the text too shallow because of this. Their evaluations also indicated if a particular question needed answering, this format was excellent. I would actually like to use more than one text but this is not a real possibility because of cost. I plan to supplement the text, as yet unselected, with some of the free online tutorials.

The students taking this course must have taken two courses involving programming require the text, *Computer Science, A Structured Approach Using C++*, by Forouzan and Gilberg, Brooks/Cole Publishing. They already know how to write programs, but do not necessarily know much about OOPs programming. In fact, some of them are quite convinced they know much more about OOPs programming than they actually do.

The system used in presenting the course was Blackboard 5 (<http://www.blackboard.com/>). This system provides email, a grade book, automatic test grading certain types of test, such as multiple choice. a assignment page, an announcement page, a document page, and a discussion forum. There are a number of other items available, but for this course, these were the main ones used. What was not provided was a simple system for preparing multiple choice tests that could be loaded directly onto the system. However, there was a free work around provided by a test translation service. The BlackBoardGenerator group describes themselves as:

A group set up to keep people informed about the status, code and improvements (or lack thereof) to the Blackboard Quiz Generator at <http://www.wcc.vccs.edu/services/blackboard>. It is not affiliated in any way with the company, but is just an enhancement that makes the product easier to use.

I found their assistance most helpful as a test could be prepared in ordinary text format, and then translated into Blackboard format. The system is in Java.

There are commercial systems for this, but our lack of money precluded even thinking about them.

The students had no problems with using the Blackboard system. They especially liked that all resources for the course were always available. And since this included the grade book, they knew at any point how many points they had accumulated. They also knew when I had completed grading (early submission was encouraged) everything send to me because they would see the announcement, "All grades posted as of this time and date.", since there was a time and date stamp automatically provided. I have yet to hear a single student complaint about it.

I did not use the digital drop-box in the BlackBoard system because I could not manipulate the mail. For example, I might want to deal with all problem five's, or all mail sent by a particular person. It is in date order and there isn't much you can do about it.

Since the common grading scheme in the United States may be unfamiliar to the reader, I will give an outline of how it works, otherwise the points for problems may make no sense.

We use the letters A, B, C, D, and F, where an A is an average of 90-100%, B is an average of 80-89%, C is an average of 70-79%, and D is an average of 60-69%, and F is 0-59%. An A is considered to be superior work, a B, excellent work, a C, average work, and a D, passing but poor. A grade of C, or better, is required in all courses in computer science and mathematics.

There were 13 programs (20 points each), 13 quizzes (10 points each) a midterm and a final, each worth 100 points. The total points available were then  $13 * 20 + 13 * 10 + 2 * 100 = 590$  points. So to receive an A, a student had to accumulate 531 points, etc.

A 10-point quiz is only worth 1.69% of the course. This was my method of making sure they read the book. It may come as no surprise that many students only look things up, rather than read the text. This worked. And if one of the quizzes was bad (I goofed it), it didn't much matter, because of their low value. I credit this with the fact that suddenly some poor students became good students. They may never have associated reading the text and their grades. I set up a discussion group for each of these quizzes in which information could be exchanged, with the stipulation they could even tell another student what page the answer was on but could not say what it was.

The midterm and final were different. Both were multiple choice examinations, the midterm being 50 questions and the final being 100 questions. Each exam was given over a four day period; that is, when I released them, students had four days to complete them. This fits in with my idea of how a distance learning course should be done – one time, I had a student in Osaka, Japan and another in Paris, France taking an exam. Asynchronous approach is what is intended. As above, I made a discussion group available for each. Some students complained it took them five hours to complete an exam. My comment was very sympathetic; that is, “We could allow only one hour for said exam.” I asked if they planned to hang me in effigy. The reply was, “Effigy was not what we had in mind!”

Since I did an item analysis on the examinations, I can say with some assurance, that there was little or no cheating. When someone works that long on a test, he or she is not likely to hand the results over to someone else.

I have little interest in testing, and have little belief in the results – to date nobody has been interested in what I could do in an hour or even two hours. So I am delighted when a student tells me that only 10 points are required for an A in the course. I want programs, not paper and pencil work (tests). Of course, designing programs on paper is necessary.

The manner in which programs were sent to me is shown by an example below. I had only to unzip them into the proper folder for processing. As mentioned above, the digital drop-box did not meet my needs, but email with a filter did.

Send to me ([Higginbotham@selu.edu](mailto:Higginbotham@selu.edu)) the following files zipped together:

CS35501.java  
CS35501.class

The zip file name is to be:

355-01-LastNameFirstname.zip

The subject line should be the name of the zip file; that is

355-01-LastNameFirstname.zip

The 01 indicated that this was problem 01. Grading a batch of 01 programs meant only had to sort on the subject line so that they were all together.

I have included a number of the problem statements, mostly those that the students said they enjoyed or found informative, which was not always the same. The programs had to be short because of the shortness of the summer semester. Also, I did not want the complexity of the program to obscure what was really intended, which was learning about OOPs programming.

Program Two (learn to use the JOptionPane class):

Input a number using the JOptionPane class from the keyboard. Convert this character string to a double precision number.

If this number is greater than 0, then raise it to the 3.1 power otherwise issue a statement that this is a number that is 0 or less, which means the pow function will not work with it.

Read and raise numbers to a power until an X (lower or upper case) is entered, at which time shut the program down.

Program Three(dealing with mathematical functions):

Write a static method (pp88+), which can be used to find the third side c of a triangle, given the sides a and b, and the include angle C.

The method declaration is to be:

```
private static double calculateOppositeSide // returns c
(
    double a,          // side a
    double b,          // side b
    double C           // angle C in decimal degrees
);
```

The main is to input the data, a, b, C, which you will assume is correct, using JOptionPane.

Output c.

Test with a = 1.0, b = 1.0, and C = 60.00 degrees, which should give c = 1.0, as this is an equilateral triangle.

If you have forgotten the Law of Cosines, it is:

$$c^2 = a^2 + b^2 - 2 * a * b * \cos C$$

Program Four ( A little more mathematical function usage -- students liked this one because so many of them are involved in one way or another with shipping. Two major US ports, New Orleans and Baton Rouge are within 100 kilometers. ).

Write a static method , which can be used to find the great circle distance between two points on earth, given the locations in degrees, minutes, and seconds.

The method declaration is to be:

```
private static double calculateGreatCircleDistance // returns distance in kilometers.
(
```

```

double latitude01
double longitude01
double latitude02
double longitude02
);

```

The main is to input the data, latitude01, longitude01, latitude02, longitude02, which you will assume is correct, in decimal degrees (Darwin is 12.463 S and 130.841 East) to the static method. The main is to obtain the original data via JOptionPane as degrees, minutes and seconds.

North is considered to be +  
 South is considered to be -  
 East is considered to be +  
 West is considered to be -

(If you think of the above as graph paper, then you can see that above the equator is + and below is negative, and that west of Greenwich, GB is - and east is +

Output the distance in any manner you wish, both in miles and kilometers.

To compute great circle distances:

Assuming Earth is a perfect sphere of radius 6367 km or 3956 mi, convert longitude and latitude to radians (multiply by pi/180), then compute as follows:

```

theta = lon2 - lon1
dist = acos(sin(lat1) × sin(lat2) + cos(lat1) × cos(lat2) × cos(theta))
if (dist < 0) dist = dist + pi
dist = dist × R

```

The resulting distance is in kilometers or miles, depending on the units of R..

Test with a number of different places, but here is one you might find interesting.

Hammond, Louisiana

Degrees	Minutes	Seconds	
30	31	15	North
90	25	03	West

Darwin, Australia

Degrees	Minutes	Seconds	
12	27	0	South
130	50	0	East

Distance between the two is 15379 km or about 9558 miles.

One mile = 1.609 km.

Be assured that if you get in a hurry on this one, it is going to bite!

Program Six(More about functions -- another one that students liked)

How many days, hours, minutes, and seconds have you been alive?

Write a program to calculate how many days, hours, minutes, and seconds you have been alive. The birth date is to be read from the keyboard in a manner you wish, the current date, etc., is to be gotten from the system, and the result is to be written in any manner you wish, assuming professional quality.

Program Nine(first applet):

Convert Four (great circle distance) into an applet, making as few changes as possible.

Program Eleven(Students liked this one – one student managed to get 16 balls in motion, and whenever one touched the other, both would disappear. )

Write an applet which drops a picture of a ball down the screen, while another one is going up the screen. The background should be yellow.

Program Thirteen(They had a good time with this one. If the timing is just right, there comes a time when the propeller appears turn in one direction and then suddenly reverse itself, which is an optical illusion.)

Write an applet which has a baton spinning like a propeller. The background should be yellow. See chapter 20 and chapter 14.

The final grade distribution was A's, 42%, B's 37%, and C's 21%, with 5 dropping the course without a grade.

Students were very nice to me when evaluation time came. The only negative thing said was they did not like the book. But as one commented, "We did it to ourselves!"

The students themselves told me the two most important things they learned in this course were the object-oriented approach, which cut down on the work required, and the use of threads in applets. This is the same reports that I have had before, so they come as no surprise. Also, they preferred writing applets over applications came as no surprise. What did come as a bit of surprise was that a number of them preferred the online course to the regular classroom.

The idea that the instructor does not get to know the students is incorrect. I actually think there is more contact – the student and instructor are dependent on each other more than in the normal setting. I cannot look over a classroom and see that I am not getting my points across. I have now taught 8 DL courses, and am teaching 3 DL courses this semester. I do enjoy this approach to education.

This has been a very informal paper concerning the teaching of OOPs programming via Java via distance learning. My belief is that online courses will become more and more part of the landscape. Here, at Southeastern Louisiana University, we have 4,000 of 15,000 students involved in DL in one form or another. They are a lot more work for the instructor, or at least they are for me. They cost the university more than the normal course, according to our Board of Regents, who have added a \$36 surcharge, which is about 10%, for a 3-hour DL course. Students tell me it takes more effort on their part. On the other hand, I have had shut ins (unable to leave their house), people whose schedules do not make it possible for them to attend class, and people who just don't learn well in a class, have an opportunity for an education.