

A JAVA VIRTUAL MACHINE LEARNING OBJECT TO SUPPORT INTRODUCTORY PROGRAMMING COMPUTING

Bernard Chalk

Faculty of Business Computing & Information
Management

London South Bank University

London SE1 0AA, UK

bernard.chalk@lsbu.ac.uk

<http://www.lsbu.ac.uk/~chalkbs>

Alan Clements

School of Computing

University of Teesside

Middlesborough, UK

a.clements@tees.ac.uk

<http://www-scm.tees.ac.uk/users/a.clements/>

ABSTRACT

This paper describes a Java Virtual Machine (JVM) learning object that was developed to support the teaching of introductory programming. Following a general introduction, the modelling of the JVM is discussed together with an overview of its main features. An evaluation procedure is then described and results presented which suggest the learning object is an effective teaching support aid. The paper concludes with a discussion of these results and suggestions for future development.

Keywords

Learning Object, Java Virtual Machine, Java programming

1. INTRODUCTION

A recent programming language survey of HE institutions in the UK revealed that Java was the most popular language for teaching introductory programming [1]. One reason for this is that Java is a freely available object-oriented language that can run on any machine having a Java Virtual Machine (JVM).

Teaching introductory programming using Java can be done either by introducing the concepts of objects, classes and inheritance at the beginning of the course (object-first approach) or introducing them after the procedural aspects of the language have been covered (object-late approach). Although the object-first approach is pedagogically sound and avoids the need for a paradigm-shift, it also creates a steeper learning curve making it incumbent upon course designers to provide support material to aid conceptualisation.

An increasingly popular way of providing learning support is through the use of learning objects. These are small chunks of interactive educational content that can be reused in different learning environments. In the area of introductory Java programming a number of different learning objects have been developed to assist students in getting to grips with the OO paradigm [2][3][4]. Some of these address a single idea or concept whereas others link several learning objects together in order to increase the richness of the learning experience. Most if not all of them use some form of visualisation technique to explain basic concepts, such as providing graphical or textual feedback as the user steps through an application or investigates a 'virtual world'. Few however offer the user a chance to substitute their own code for that 'hardwired' by the developer and none take advantage of the JVM itself as a vehicle for improving the student's ability to construct mental models.

This paper describes a learning object that was designed to support the teaching of introductory Java by allowing students to visualise the creation and manipulation of objects inside a JVM during program execution. Following a rationale for the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2006 Higher Education Academy

Subject Centre for Information and Computer Sciences

model chosen to represent the JVM, we describe a scenario in which a user single-steps through parts of a demonstration program, highlighting key features and options of the learning object along the way. We then present the results of an evaluation exercise that was carried out in an attempt to measure its effectiveness and conclude with a discussion of our findings and suggestions for future development.

2. MODELLING THE JVM

The JVM specification describes an abstract computing machine [5]. Like a real machine, the JVM has an instruction set and manipulates various memory areas at run time. It knows nothing of the Java programming language only of a particular binary format; the class file format. Java programs are compiled into class files containing instructions (bytecodes), a symbol table and some other ancillary information. When a JVM is started, various data areas are created including a *heap*, *method area* and *stack*. The heap is used to store objects that can take the form of class instances and arrays while the method area is used to store class information such as constants and the code for methods and constructors. Each JVM thread has its own private stack that is used to store frames containing local variables, operands and return values for methods.

To support introductory programming by modelling an implementation of the JVM, we had to give careful consideration to the views that would benefit conceptualisation. Clearly the heap was essential for visualising objects and parts of the method area were important for understanding the relationship between objects and classes but did we need to model the stack? Although this would help students gain a deeper understanding of parameter passing and stack traces would it reduce the cohesion of the views and clutter the user interface? On the other hand, if our model ignored it. Especially the concept of bytecodes. How would students appreciate the difference between a .java and a .class file ?. These together with the issue of how best to represent objects and classes on the graphical user interface required careful consideration.

After some experimentation with throwaway prototypes, it was eventually decided to model the JVM in terms of the heap, method area and a third byte code area in which the bytecodes corresponding to each high level programming language statement could be displayed. We also decided that because the Unified Modelling Language(UML) [6] was so widely used with the

more popular introductory development environments and text books, that we would represent our objects and classes using this notation as far as possible.

3. LEARNING OBJECT OVERVIEW

The JVM application and user guide can be found at <http://sos-tutors.co.uk/jvmrlo>, where instructions for installing the application are given. After starting the JVM learning object, users have the option of either running their own application or installing one of the in-built demonstration applications. In this section we will outline its main features by describing a scenario in which the DVDHire demonstration application is used. This models a simple DVD Hire Shop by creating and modifying instances of the classes Customer, Rental and DVD. The DVD class extends a further abstract class called RentableItem.

After installing and compiling the demonstration, the user can either run it and display the output or single-step through it and observe one or more areas of the JVM. Figure 1 illustrates the JVM after the user has chosen to view the heap in single-step mode using the Run and View menus.

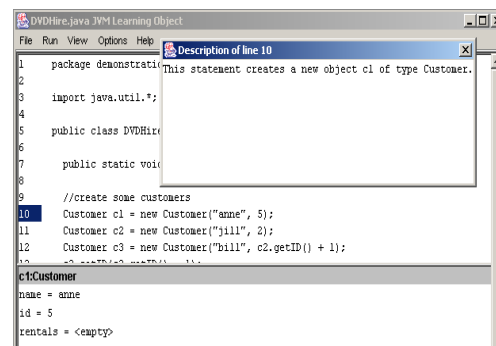


Figure 1 The JVM in single-step mode showing the heap area at the bottom

The blue square cursor in the code area near the centre indicates that the user has stepped to line 10 in the program and the heap area beneath shows the object that has been created. In this case, a Customer object c1 has been added to the heap with its name and identifier fields initialised to those shown in the constructor and with the rentals list marked empty. Because the user has chosen 'display feedback' from the Options menu, a brief description of what the program statement does is also provided in a pop-up dialogue box.

To see details of the class to which this object belongs, the method area can be opened using the View menu, as shown in figure 2.

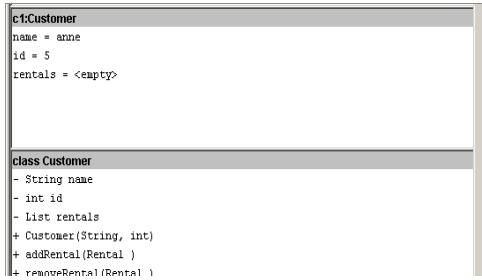


Figure 2 Heap area and method area views of a Customer object

Here beneath the heap we can see the Customer class has been loaded and that it has three private fields, a single public constructor together with several public methods such as addRental(). A scrollbar adjacent to the method area but out of view in this screen shot allows the user to see these other methods.

Figure 3 illustrates the creation of another object d1 of type DVD, which appears as the user steps to line 15. This time the method area shows that class has three public static fields (in bold) together with a two private instance fields.

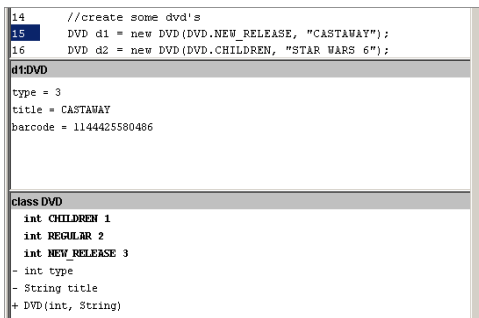


Figure 3 Representing static and instance fields of a class

As mentioned earlier, this class extends the RentableItem class which has a field called barcode and a couple of methods for setting and getting its value. These cannot be seen in the current view of the method area, but had the user used the Options menu to select 'show superclass details' then we would have seen this additional information highlighted in blue, as shown in figure 4.

Fields and methods of higher-level superclasses such as Object are also displayed but in light grey.



Figure 4 Viewing superclass fields and methods

When a method is invoked upon an object the heap area gives a view of the object in its new state, as shown in figure 5. If the user also happens to have selected to view the byte code area, the JVM instructions for this method are also displayed in the byte code view shown beneath the method area.

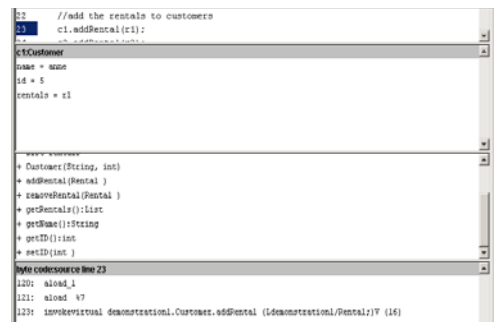


Figure 5 Byte code instructions for method addRental()

If an assignment statement is encountered, such as that shown in figure 6, then the references are recorded in a line-bordered list at the bottom of the object.

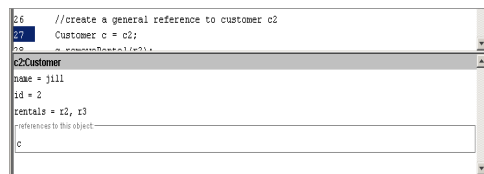


Figure 6 Representing object references

This allows the user to see that they haven't created a new object c but have a reference to an existing object, in this case, c2 so that any changes that are now made to c2 will be reflected in c.

When loop statements such as while and for loops are encountered in the application, the current version of the JVM learning object only displays the state of objects during the final pass and not for each iteration. In a similar way when selection statements such as switch or if are encountered,

objects are only displayed if the particular line of code is reachable, otherwise the heap and other areas shown nothing.

The application or any of its dependant classes can be modified at any time using the File -> Edit menu, provided the user exits the stepping mode first. The application can also be closed and a new one started, provided it exists in a subdirectory of the same working directory specified when the JVM is first started. This gives the learning object great flexibility and allows it to be used in a wide range of learning contexts.

4. EVALUATION

To measure the effectiveness of the learning object a group of 10 student volunteers were asked to undertake a short multiple-choice test before and after using the JVM learning object to perform a simple simulation exercise. The same test was used in both cases and all students had some understanding of the Java programming language although none of them had any understanding of the Java Virtual Machine. Details of the test and the simulation exercise can be found on our website <http://sos-tutors.co.uk/jvmrlo>.

The test consisted of 15 questions, 7 designed to test their understanding of objects, 6 to test their understanding of classes and 2 to test their understanding of byte codes. After each question the students were asked to rate the confidence in their answer as either certain, confident or unsure.

The average test score results for each question type before and after using the learning object are shown in figure 7. These indicate an improvement in the average test scores for those questions concerned with objects with little or no improvement for those concerned with classes or byte codes. By contrast, figure 8 shows the average confidence level the students' had in their answers before and after using the learning which shows a significant increase of over 50% across all types.

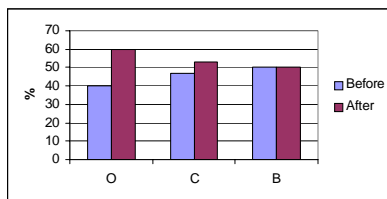


Figure 7 Average test scores by question type (O = object, C = Class, B = byte codes)

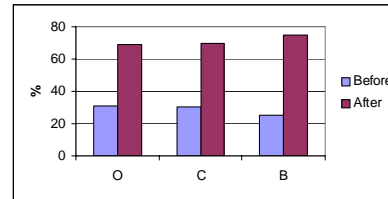


Figure 8 Confidence levels by question type before and after using the learning object

As far as usability was concerned, no formal attempt was made to evaluate this although informal interviews with the volunteers immediately after the evaluation exercise suggested that the user interface was easy to use and that the representation of objects and classes were easy to understand.

5. DISCUSSION

This paper has described a Java Virtual Machine learning object designed to support introductory programming and outlined how it can be used. The results of an evaluation exercise using a small cohort of experienced Java users have shown that using the learning object can increase the users understanding of objects but appears to make little difference to their understanding of classes, inheritance or the byte codes executed by a JVM. It has also shown that using the learning object increases the users general confidence and perceived understanding.

As with any small scale evaluation exercise, the results could be influenced by a number of factors including the way the simulation exercise was carried out, the structure and type of questions used in the evaluation instrument and the students attitude to computer based learning. Whether novice programmers would have a similar learning experience needs a more detailed investigation and requires the use of a broader range of evaluation instruments

6. FURTHER DEVELOPMENT

Although the current version of the learning object meets its original design objectives, there are several limitations and directions in which further development is needed. One is its inability to show the creation of objects other than by using the *new* keyword or by *cloning* another object. Other methods such as invoking `newInstance()` on a Class

object or by deserializing an object cannot be used, although on novice programming courses for which the learning object was designed, this is unlikely to be a major problem. Another is its inability to allow tutors to modify the commentary to suit the instructional methodology, which is a fundamental aspect of learning object reusability. These together with other related issues would therefore need to be addressed in any future version of the JVM learning object.

7. ACKNOWLEDGEMENTS

We wish to thank the Higher Education Academy for supporting this project under a development funded grant 2005/06.

8. REFERENCES

- [1] Chalk B. and Fraser K., *A Survey on the teaching of introductory programming in Higher Education*, JICC 10, London Metropolitan University, (2006)
- [2] Boyle T., *Design Principles For Authoring Dynamic, Reusable Learning Objects*. Procs. of the 19th ASCILITE Conference, Auckland, New Zealand, 2002
- [3] Chalk P., Bradlet C. and Pickard P., *Designing and Evaluating Learning Objects for Introductory Programming Education*, ITiCSE, 2003
- [4] Ford L., *A Learning Object Generator for Programming*, ITiCSE, 2004
- [5] Lindholm S. and Yellin F., *The Java Virtual Machine Specification 2nd Ed*, Addison-Wesley, 2003
- [6] Booch G., Rumbaugh J., Jacobson I., *The Unified Modelling Language User Guide*, Addison-Wesley, 2001