# Introducing Java to the IT Master's Curriculum

**Wendy Lucas**
*Bentley College, USA*

## INTRODUCTION

The object-oriented programming paradigm has gained popularity in both industry and academia, and Java is becoming the language of choice. Yet, it can be a difficult language to learn, with many hurdles for novice programmers. This overview describes how Java was successfully introduced as the first programming language in an information technology master's program at Bentley College. Careful consideration was given to a variety of factors, including when to introduce object-oriented concepts, which integrated development environment to use, and how to support students with minimal prior programming experience. The impact of these choices on the learning experience and the factors that led to the successful implementation of Java as a first programming language are described.

## BACKGROUND

The Java programming language was developed at Sun Microsystems in 1991 for use in consumer electronics devices, such as television sets and VCRs. It, therefore, needed to be both small and portable. While the language has grown in size with each new release, the Java Virtual Machine continues to ensure implementation-independent code that can be run under a variety of operating systems. Coupling this capability with the ability to run Java applets from Web pages was what first attracted interest in the language when it was initially released in 1995. Since that time, Java has developed into a general-purpose language used throughout enterprise-wide distributed applications.

While Java has gained acceptance and widespread use in industry, it has also made inroads into academia. It is now taught extensively in intermediate programming courses and has been widely adopted as the first programming language. Its inclusion in introductory courses, however, has been problematic due to inherent complexities in the language. In comparing Java to C++, for example, it soon becomes clear that many of Java's "simplifications" are not correlated with a simpler learning experience for beginning programmers. Although Java does not support multiple inheritance, a difficult concept for students learning C++, it does allow classes to extend multiple interfaces, which is similar in purpose and complexity. Java does not permit pointer arithmetic and hides pointers from the user, but beginning programmers must understand the concept of references in order to work with objects and arrays. Other problems include: the library documentation is often ambiguous; the encapsulation model is actually more complicated than that of C++; and a large number of methods in the class library throw exceptions that must be caught or passed to a caller. Benander, Benander, and Lin (2003) found that, while professionally employed programmers understand the importance of catching exceptions, many students fail to appreciate Java's exception handling approach. Even capturing keyboard input from the user is difficult, as Java does not provide basic support for such input in non-GUI programs. As a result, authors typically provide their own methods, requiring students to develop a basic understanding of packages, classes, and methods at a very early stage (see, for example, Lewis & Loftus, 2003; Savitch, 2004). One way in which Java is truly simpler than C++ is in providing automatic memory management in the form of a garbage collector.

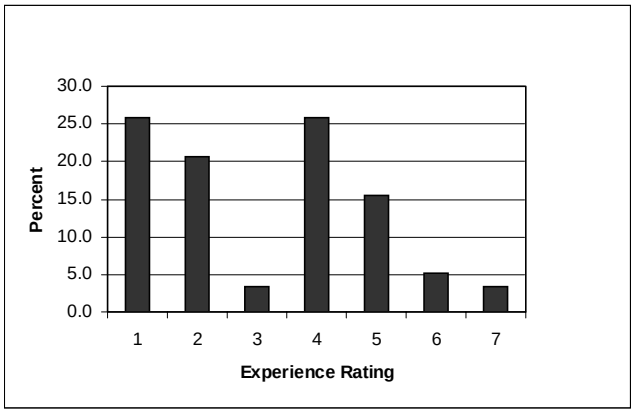
Given the difficulties new programmers must overcome in learning Java, some educators, such as Collins (2002), have reached the conclusion that it is not reasonable to expose students to this language in their first programming course. Others have sought means for shielding students from some of Java's complexities. Roberts (2001) describes the use of the MiniJava environment, which contains a subset of the standard Java release along with simplifying enhancements that make it easier to use. Another development environment, BlueJ, was specifically designed with teaching in mind (Poplawski, 2001; Sanders, Heeler & Spradling, 2001). It provides an easy-to-use interface with customizable templates for class skeletons, and allows the user to instantiate objects and test methods without having to write a driver program. Kölling and Rosenberg (2001) used this environment during three semesters of an introductory object-oriented programming course. The system supports an "objects first" approach, provides visual representations that help students understand the relationship between classes and objects, and supports student experimentation, thereby promoting frequent and early testing of

code. Lewis and Watkins (2001) also found that BlueJ helped them explain and demonstrate object-oriented concepts from the beginning of their introductory programming course within the MSc in Computing program, in which the majority of students were graduates from other disciplines. Barnes (2002) also described an objects-early approach to teaching introductory Java through the use of LEGO® MINDSTORMS™ kits. Providing physical models was found to enhance and support the introductory programming course experience.

The debate between following an object-oriented or procedural paradigm when first teaching Java has persisted from the earliest days of its inclusion in programming courses. Results of a survey of sixty-one students who had taken or were taking a Java programming course showed that while object-based programming was not considered particularly difficult to learn, object-oriented programming concepts were difficult (Madden & Chambers, 2002). A recurring objection to "objects first" is that students must proceed on faith that the concepts and structures, which they make use of early on, will be explained to them and understood later in the course. Based on the premise that students should gradually build upward from primitive data types and control structures in order to free them from the immediate conceptual load associated with abstract data types, Cecchi, Crescenzi, and Innocenti (2003) follow a "structured programming before object-oriented programming" approach in their CS1 course. An automated development tool called JavaMM was used in this course to remove the burden of creating the complex structure of Java programs from the student. A preliminary evaluation indicated that the tool was very useful in improving the success rate of students.

Little research exists on the use of Java as a first programming language in graduate information systems (IS) or information technology (IT) programs. While these programs face a number of the same issues as those mentioned earlier, there are significant differences between IT graduate and undergraduate students. For the former, their first programming course may well be the only one they take. In addition, many of them expect to see environments and applications similar to those they have been exposed to in the workplace. They may also have a broader range of prior programming experience. It was necessary to take these differences into account when designing the course described next.

*Figure 1. Self-ranking of prior programming experience*




## OBJECT-ORIENTED PROGRAMMING COURSE

The graduate level object-oriented programming course within the master's level IT program was taught for the first time in the fall of 2001 to 58 graduate students in three sections. On a scale from one (*novice/beginner*) to seven (*expert*), the average student prior programming experience level at that time was $3.14 \pm 1.78$. Figure 1 shows the distribution.

A proactive approach is taken to support students in what, for many, is their first exposure to programming. The following are descriptions of choices that have been made for enhancing the learning experience without adding to student anxiety over learning to program in Java.

### Supporting Materials

One critical decision concerned which integrated development environment (IDE) to use. After extensive trials with a number of environments, the Borland® JBuilder IDE was selected due to the strength of its built-in help facilities and its tools for writing, running, and debugging code. While it could be confusing at first, JBuilder is easier to use than most of the professional environments found in the industry (Savitch, 2001). Plus, several students had expressed interest in working with JBuilder because it was being used in their own work environments. JBuilder was installed on the computers in all of the technology classrooms and computer labs, and students were encouraged to download their own copies from the Web.

A review of several textbooks led to the selection of *Computing with Java: Programs, Objects, and Graphics* by Gittleman (2001). This book was chosen for its in-depth coverage of critical course concepts, the clarity of its text, and its frequent and easy to follow code examples. The course Web site provides access to weekly lecture notes, assignments, individualized grades and comments, source code for programming examples from the lecture, and links to other relevant sites, such as Java's class libraries.