Teaching Object-Oriented Programming: A Comparison Of Java And Objective-C

Alexis Koster, San Diego State University, USA

Abstract

The programming language Java has been for many years the language in which many Web applications as well as large server applications have been developed. More recently, it has also been used in the development of Android applications. It has often been adopted as the primary teaching language in both introductory and advanced programming courses. Due to its use on the iPhone and the iPad, Objective-C is gaining popularity and is now taught in some programming courses. Not as well designed as Java and not as general-purpose as Java, Objective-C is unlikely to supplant it in college courses.

Keywords: Java; Objective-C; Object-oriented programming; Data encapsulation; Inheritance

INTRODUCTION: INFORMATION HIDING

Programmers involved in the design and programming of large projects should know as little as possible about the internals of the modules written by other programmers; this was demonstrated in a seminal experiment in software engineering (Parnas, 1972). In particular, the implementation of the data in a module must be hidden. This principle is known as "information hiding", and is closely related to the concepts of data abstraction (Liskov, 2001) and data encapsulation (Poo et al., 2008). At the time of Parnas' experiment, programming languages had features that went against this principle. For example, Fortran has a feature called *COMMON* making data available to all modules. So does the programming language C, with the global *extern* variables (which also exist in Objective-C).

One of the main objectives of the introduction of classes and objects in programming languages was precisely to provide a general mechanism for information hiding. Classes and objects are now required features of languages taught in programming courses (Koster, 2010). The object-oriented languages in use today include, among others, C++, C#, Java, Objective-C, and Visual Basic. The Java language has been for many years the language of choice in computer science departments. Objective-C (Kochan, 2012), an older language than Java, has recently seen an increase in its teaching.

Java was introduced by Sun Computer Systems in 1995 as a language for the development of Web applications. An essential factor in Java's acceptance and popularity stems from the way Java programs are processed using the Java Virtual Machine (Poo et al., 2008), making Java a multi-platform language. A second reason for the popularity of Java stemmed of the availability of a huge library of classes, many of which are useful for the development of GUI-based applications

At the time Java made its initial appearance on the software scene, the language of choice for developing large system applications was C++. C++ had taken the full C programming language as its basis and added to it object-oriented features (Stroustrup, 1997). Java took as its basis a small and clean subset of C and integrated object-oriented features in that subset. As a consequence, Java supplanted C++ in many software development applications because it was much cleaner and simpler than C++.

Objective-C, designed in the early 1980's, was adopted by Apple in 1996. It became the language of development of applications under the Mac OS X operating system, and under the iOS operating systems of the iPhone and the iPad. This led to the growing popularity of Objective-C and to the increase of its teaching in programming classes. As for C++, the full C language is a subset of Objective-C, which added to it object-oriented features, including its class libraries known as the Foundation framework, Cocoa, and Cocoa Touch.

CLASSES, OBJECTS, METHODS

Classes and objects are the building blocks that realize the principles of information hiding and data encapsulation. Another important feature is the use of hierarchies of classes, which provide for re-use of software. In a nutshell, when a class has been created (for example class BankAccount), programmer Jane using that class does not need to know how data are represented (they are "private" data), and does not need to know how the methods (such as the method creditDeposit and print_Statement) are implemented. She needs to know how to invoke those methods (for each method, its name, the number of parameters and their types, and the type of the data returned by the method). Jane can create objects of the class BankAccount such as tom_Account and invoke the "public" methods of the class, such as creditDeposit to process tom_Account. Moreover, if Jane works for a bank that has a special type of bank account, for example a savings account, Jane can create a subclass of BankAccount called SavingsAccount with its own methods (for example the method compute_Interest), whereas the existing methods of the superclass BankAccount may still be used by objects of the subclass when needed (inheritance). Both Java and Objective-C have the needed programming features for those concepts, but they do it in different ways. Here are a few of these differences.

Reference To Objects

Java has introduced a simple and clean feature whereas Objective-C uses the C-based feature of explicit pointer, which is more complex to express and more difficult for students to understand. Here is how we would declare tom_Account to be a variable that will refer to an object of the class BankAccount in Java and in Objective-C;

```
BankAccount tom_Account;          (Java)
Bank_Account  *tom_Account;       (Objective-C)
```

The asterisk in Objective-C comes directly from the language C. It indicates that tom_Account will contain a pointer to an object of the class BankAccount, as pointers are explicit elements of the language, which can be processed on their own. Forgetting the asterisk is a very common error.

Methods and functions

Java has one unique syntax for all methods and functions. Continuing the previous example, here is how we would create an object of the class BankAccount in Java, set the initial balance of $1000.0, and credit a deposit of $200.0:

```
tom_Account = new BankAccount();
tom_Account.setInitialBalance(1000.0);
tom_Account.creditDeposit(200.0);
```

The method BankAccount() in the first statement is called a *constructor*. It is used to create and initialize an object of the class BankAccount; setInitialBalance and creditDeposit are two methods of the class BankAccount. As *public* methods, they are part of the interface of the class BankAccount.

In Objective-C, the similar statements would be:

```
tom_Account = [[BankAccount alloc] init];
[tom_Account setInitialBalance:1000.0];
[tom_Account creditDeposit:200.0];
```

As seen in those statements, Objective-C has an unusual syntax for indicating the arguments of a method. Note also that Objective-C uses a different syntax for functions that are not methods (such as *NSLog* to display to the console or the *main* function), creating possible confusion for students as well as more complexity.

Information Hiding

In a Java class, the keyword *private* indicates that the data (also called "fields" or "instance variables") of an object of that class are hidden from other classes. We would start the definition of the class BankAccount as follows:

```
public class BankAccount
{
   private Owner customer;
   private double initialBalance;
   private double endBalance;
   …….
```

Declaring instance variables as *private* is strongly recommended, but Java also provides *protected* instance variables, which make them accessible to subclasses; package-access instance variables, which make them accessible to classes of the same package; and *public* instance variables.

Things are much more muddled in Objective-C. The methods of an Objective-C class are *declared* in a section called the *interface* section, they are *defined* in a section called the *implementation* section. Instance variables declared in the interface section are accessible to the class and its subclasses, but hidden from other classes. Instance variables declared in the implementation section are completely private (not even accessible by the subclasses). It is considered good practice to give the name of an instance variable to the method that retrieves its value. For example, consider part of the implementation section for the class BankAccount.

```
@implementation BankAccount   (Objective-C)
{ ……….
  double initialBalance
}
……….
-(double) initialBalance
{
   return initialBalance;
}
```

In the main program, we would retrieve the intialBalance of tom_Account with the following expression:

```
[tom_Account  initialBalance]
```

where initialBalance is the method, not the variable. This is very confusing for students and, in some sense, seems to be contrary to the concept of information hiding. Objective-C goes even further. Instead of declaring the instance variable initialBalance, the method initialBalance, and the method setInitialBalance (which sets the value of the instance variable), we can do as follows, using the *property* construct.

1) In the interface section, have the following statement:
   @property double initialBalance;
2) In the implementation section, have the following statement:
   @synthesize initialBalance;

Then Objective-C internally generates the instance variable initialBalance and the accessors method for it (the method that gets its value and the method that sets it). At this point, the dot notation should be used, for those accessor methods, for example

```
tom_Account.initialBalance = 1000;
double newBalance = tom_Account.initialBalance;
```

Again initialBalance in both of the statements is not the instance variable, even if it looks like we are using the name of the instance variable (in particular in the first statement), seemingly violating the principle of information hiding.

Because of the confusion created by this use of properties, it is now advised (Kochan, 2012) to use a slightly different name for the variable and the property, starting with an underscore. The statement

```
@synthesize initialBalance = _intialBalance;
```
indicates that the name of the instance variable is _initialBalance.

COURSE OUTCOMES

In Spring 2013, 22 students took the Objective-C course. In Spring 2014, 27 students took the Java course. Both courses are Management Information Systems elective courses. Senior students and a few graduate students enroll in them. Students already took a Visual Basic course, which is a prerequisite for those two courses. For each of the Java course and the Objective-C course, the first examination included eight very basic questions about the topics discussed above. For the Java examination, there were 26 errors overall for the 8 questions, or 0.96 error per student. For the Objective-C course, the 22 students totaled 56 errors for the 8 questions, or 2.54 errors per student. Although those figures are not produced by a rigorous experiment, they seem nevertheless to indicate that students understand better the basic concepts of object-oriented programming after a Java course than after an Objective-C course.

CONCLUSIONS

This comparison between a few basic features of Java and Objective-C shows that, overall, Java is a better designed programming language than Objective-C. It is better suited to teach the concepts of object-oriented programming, in particular of information hiding. Results of course examinations support this assertion. Java large industry use makes it a more desirable choice for students. It is unlikely that Objective-C will overcome Java in the classroom.

AUTHOR INFORMATION

Alexis Koster
After working several years in industry as a software engineer, Alexis Koster joined the MIS Department at SDSU in 1983, where he has been teaching database systems and application development with various languages, including Java and Objective-C. His current research interests focus on the role of the Internet in changing the music industry and on the use of object-oriented programming in application development.
E-mail: akoster@mail.sdsu.edu

REFERENCES

1. Kerrigan, B. & Richtie, D. (1988). *The C Programming Language, 2e*. Upper Saddle River, NJ: Prentice-Hall.
2. Kochan, S. (2012). *Programming in Objective-C, 4e.* Upper Saddle River, NJ: Pearson.
3. Koster, A. (2010). Are Academic Programs Adequate for the Software Profession? *The American Journal of Business Education,* March.

4. Liskov , B. & Guttag, J. (2001*). Program Development in Java – Abstraction, Specification, and Object-Oriented Design.* Reading, MA: Addison-Wesley.
5. Parnas, D. (1972). Some Conclusions from an Experiment in Software Engineering Techniques. AFIPS Conference Proceedings. Paper presented at the Fall Joint Computer Conference, Anaheim, Ca, 5-7 December. Montvalle, NJ; AFIPS Press.
6. Poo, D., Kiong, D., & Ashok, S. *Object-Oriented Programming and Java, 2e.* New York, NY: Springer.
7. Stroustrup, B. (1997). *The C++ Programming Language, 3e*. Reading, MA: Addison-Wesley.

NOTES