Structured programming in Java

Jeremy Gibbons
School of Computing and Mathematical Sciences, Oxford Brookes University, Gipsy Lane, Headington, Oxford OX3 0BP, UK. jgibbons@brookes.ac.uk. [*]

The determined Real Programmer can write
Fortran programs in any language.[5]

Abstract

We argue that for computing majors, it is better to use a 'why' approach to teaching programming than a 'how' approach; this involves (among other things) teaching structured programming before progressing to higher-level styles such as object-oriented programming. We also argue that, once it has been decided to teach structured programming, Java is a reasonable language to choose for doing so.

Keywords: Java, structured programming, object-oriented programming, teaching.

1 Introduction

Last year at this workshop, McLaughlin[3] presented a paper entitled *Oh, by the way: Java is object-oriented.* He observed that many institutions were switching to Java as a first programming language, without capitalizing on the object-oriented (OO) features of Java. McLaughlin argued that this was a Bad Thing: 'if Java becomes the core language then OO must be taught as the central theme of the development curriculum'.

In this paper, I would like to present the opposite point of view. I submit that there is an argument for teaching structured programming (SP) before teaching OOP, instead of teaching OOP from the start; moreover, I submit that Java is a good choice of language in which to teach SP. The remainder of this paper will present those two arguments—the first briefly, since it is somewhat off-topic for this forum, but the second in more depth.

2 Structured programming before object-oriented programming

There is no doubt that the principal features of OOP (namely encapsulation of related data and code into objects, and inheritance in a hierarchy of object classes) are of great benefit when it comes to constructing large software systems[4]. However, it is a long journey from the first programming course to the construction of large software systems.

Approaches to teaching programming divide essentially into two styles. One style starts at a high level of abstraction, typically using a pure functional language for the first programming course[2]. This has the advantage of enforcing good habits, but the disadvantage of not illustrating why those habits are good. This is very much a 'how' approach, showing students best practice from the start.

---

[*] Technical Report CMS-TR-98-02, School of Computing and Mathematical Sciences, Oxford Brookes University. Presented at *2nd Conference on Java in the Computing Curriculum,* South Bank University, London, January 1998, and to appear in *SIGPLAN Notices*.

The other style starts at a low level of abstraction, typically with an introduction to computer architecture, then makes brief forays into machine-language and assembly-language programming before working up to a third-generation language and beyond. This approach has the disadvantage of exposing students to all sorts of grotty details, but on the other hand students gain a personal appreciation of the benefits of the various innovations in programming. One might characterize this as a 'why' approach, revealing the problems of earlier styles of programming to motivate the move to higher levels of abstraction.

We claim that the 'why' approach is the more appropriate for computing majors: it is these students who should have an appreciation of why we do things the way we do, and it is these students that we have for long enough to do justice to this approach. In contrast, the 'how' approach is most suitable for non-majors, who are more interested in programming as a tool than as an end in itself, and who cannot usually afford the luxury of a thorough historical survey.

We argue, then, that it is better to introduce students, especially computing majors, to SP before progressing on to OOP and other approaches that are beneficial for programming in the large.

3 Java as a language for structured programming

We now assume that a decision has been made to teach SP, and that a programming language has to be chosen as a vehicle for teaching it. We argue that Java is a good—perhaps the best—choice currently available.

Of course, the main criteria on which to base the choice are *simplicity* and *safety*. Java is quite a small language; as much of the functionality as possible has been put into the APIs instead of the language itself. These APIs need hardly be discussed in a first programming course, so the language can be covered quickly, leaving sufficient time to concentrate on the more important topic of program design. Moreover, Java is type-safe, as all modern languages should be and in contrast to C, catching certain kinds of error at compile time.

Another advantage of Java is that it is not a dead-end. It is a fully-fledged object-oriented language, and so can be used also for a second programming course, a course on OOP, on data structures, and so on. It comes with the aforementioned APIs, providing many useful libraries for I/O, network programming, graphics, GUIs and the like. In contrast, languages like PASCAL and MODULA-2 have to be abandoned sooner or later in favour of more fully-featured languages. (On the other hand, it could be argued that teaching students several languages is a good thing.)

These first two advantages are *intrinsic* characteristics of the language; there are other advantages that are *extrinsic*, that is, accidental rather than inherent. One such advantage of Java over most programming languages is that it is freely available; Sun's JDK costs nothing, and is freely available from Sun for Solaris and Windows 95/NT and from third-party sources for Linux, Macintosh and so on. The compiler `javac` from the JDK is actually quite user-friendly. It gives helpful and comprehensible error messages, usually in roughly the right place. It is certainly better than some of the competition. (On the other hand, the debugger `jdb` is rather flaky: for a better environment, one has to pay.)

Moreover, Java is portable: all these many implementations are (at least in principal) equivalent. Most students now have their own computers, and all may soon be required to do so; they can alternate between programming at home and at the university without having to perform major surgery on their programs every time they do so.

Finally, Java is a very trendy language. It is a long time since a new software product has created such interest both inside and outside academia. This aspect should not hold much sway, but all other things being equal it has to count in favour of Java.

4 Disadvantages of Java

Java is not without its disadvantages. Some of these are intrinsic disadvantages, consequences of it being an object-oriented language. For example, short programs consisting of just a `main` method all have some variation on the structure

```
public class MyProgram {
   public static void main (String args[]) {
      ...
   }
}
```

This incantation is very 'noisy', compared with the equivalent construction PASCAL or MODULA-2. It is several weeks until students learn that `main` is a procedure, and that programs may contain several procedures. Another example is the distinction between primitive types such as `int`, which are passed as parameters by value, and reference types such as arrays, which are accessed by indirection and passed by reference; it would be nice not to have to make this distinction in a first programming course.

Another disadvantage is keyboard input, which is rather complex in Java. For example, to read an integer value from the keyboard, one must first declare an input stream and attach it to the keyboard

```
BufferedReader kbd =
      new BufferedReader(new InputStreamReader (System.in));
```

and then use the expression

```
Integer.valueOf(kbd.readLine()).intValue()
```

Moreover, Java is very careful about exceptions; this is generally a good point, but in this instance it means that any method that performs input must declare that it `throws IOException`. (The alternative is to catch the exception in the method, but exception handling is beyond the scope of a first programming course.) This second disadvantage can be remedied by providing a wrapper for keyboard input, giving simple names for common operations and catching the exceptions internally. This is the approach taken by the textbook *Java Gently*[1], and it is a good one.

A third kind of disadvantage is the sometimes odd syntax that Java has inherited from C. Parentheses around the guards of `if` and `while` statements are not necessary, at least for

preventing ambiguity. It is unfortunate that '=' is used for assignment rather than equality. At least Java has removed the '`if (n=0) ...`' pitfall of C, but this problem still persists with booleans. Perhaps worst of all is the syntactic ambiguity of `if ... else` statements, a problem shared with PASCAL but not MODULA-2: in

```
if (...) if (...) ...; else ...;
```

it is not clear whether the `else` clause matches the first or the second `if`. A related pitfall is the following, which does not do what the indentation suggests it should (the loop body consists simply of '`sum = sum+i;`'):

```
i = 0; while (i < n)
  sum = sum+i; i = i+1;
```

The above disadvantages are all intrinsic; an extrinsic disadvantage is the other side of the trendiness coin. Java is still new, and there is a distinct shortage of good textbooks on the language (*Java Gently* being a notable exception). Moreover, Java is still developing quickly—it took only a year and a half for Java 1.0 to be superseded by Java 1.1, and already Java 1.2 is on the way—and it is a nuisance to have to rewrite teaching materials every time the language changes. A language for teaching, especially a first programming language, should be stable.

5 Conclusion

We have argued that it makes sense to teach structured programming before teaching object-oriented programming, and that Java is a good language in which to do so, despite the disadvantages discussed above. At Oxford Brookes we are currently using Java as the first programming language on an MSc conversion degree (in defiance of some of the arguments about non-majors presented above). The course has run smoothly so far, but it is still in its first iteration, so it is too soon to see what the students make of it as a whole.

6 Acknowledgements

I am grateful to the participants of the *Second Java in the Computing Curriculum* conference, several of whom (particularly Fintan Culwin and Peter Burton) made comments that have helped my clarify my thoughts on the matters discussed here—even if not everyone agreed with me! I would also like to thank the students on Module 8792, who have patiently acted as guinea pigs for me to experiment upon.

7 References

1 Judy Bishop, *Java Gently*. Prentice-Hall, 1996.
2 Tim Lambert, Peter Lindsay and Ken Robinson, *Using Miranda as a First Programming Language*, Journal of Functional Programming 3(1) 1993, p5–34.
3 Phil McLaughlin, *Oh, by the way, Java is object-oriented…* Java in the Computing Curriculum, South Bank University, February 1997.
4 Bertrand Meyer, *Object-oriented software construction*. Prentice-Hall, 1988.
5 Ed Post. *Real Programmers don't use PASCAL*. Datamation, July 1983, p263–265.

Appendix: Structured programming features in Java

- the 'Hello World' program

```
public class HelloWorld {
  public static void main (String args[]) {
    System.out.println("Hello, World!");
  }
}
```

- variables and output

```
int fahrenheit = 451;
int celsius;
celsius = (fahrenheit-32) * 9/5;
System.out.println(fahrenheit + "oF is " +
  celsius + "oC");
```

- input (using wrapper class)

```
number = kbdIO.readInt("Enter integer: ");
```

- selection

```
switch (month) {
  case 9: case 4: case 6: case 11:
    monthLength = 30; break;
  case 2:
    if (year % 4 == 0)
      monthLength = 29;
    else
      monthLength = 28;
    break;
  default:
    monthLength = 31;
}
```

- iteration

```
int nPrimes = 10, primesFound=0, cand=1;
while (primesFound<nPrimes) {
  cand++;
  boolean isPrime = true;
  for (int divisor=2; divisor < cand; divisor++)
    if (cand % divisor == 0) isPrime = false;
  if (isPrime) primesFound++;
}
```

- procedures and functions ('class methods' in Java)

```
static int fact (int n) {
  if (n == 0)
    return 1;
  else
    return n * fact(n-1);
}
```

- arrays

```
static void sort (int[] a) {
  for (int i=0; i<a.length; i++) {
    int min=i;
    for (int j=i+1; j<a.length; j++)
      if (a[j]<a[min]) min=j;
    int t = a[i];
    a[i] = a[min];
    a[min] = t;
  }
}
```

- records

```
class Student {
  String name;
  int grade;
}
...
Student fred = new Student();
fred.name = "Fred Bloggs";
fred.grade = 75;
```

- linked lists

```
class ListNode {
  int data;
  ListNode next;
}
```