Problems in the Initial Teaching of Programming using Java: The case for replacing J2SE with J2ME

Ian Utting
Computing Laboratory
University of Kent at Canterbury
Canterbury, Kent, UK
+44 1227 823811

I.A.Utting@kent.ac.uk

ABSTRACT

In their analysis of the use of Java as a first teaching language, the ACM Java Task Force (JTF) identified a number of issues with the Java language and APIs which caused significant pedagogic problems. The focus of their work, and hence of their characterisation of the issues, was the Java "Standard Edition" (J2SE).

This paper contends that the version of Java designed for programming small devices (Java 2 Micro Edition, J2ME) does not suffer from these problems identified by the JTF to the extent that the (more familiar) J2SE does, and suggests a number of other reasons why J2ME represents a good choice as a first programming language.

Categories and Subject Descriptors

K.3.2 [**Computers and Education**]: Computer and Information Science Education – Computer Science Education

General Terms

Languages.

Keywords

Introductory programming, programming languages.

1. INTRODUCTION

In its "Taxonomy of Problems in Teaching Java" [1], the ACM Java Task Force (JTF) identified from the literature a number of problems which have been observed in the use of Java as a first teaching language. Setting aside those issues which had been resolved (or at least addressed) since they were first identified, the JTF is engaged in addressing the remaining issues by a combination of constructing a restricted view of the Java Standard Edition API documentation, and by producing a number of new, pedagogically-focused APIs.

In this paper, I examine the practicality of an alternative solution: replacing the use of the familiar "Standard Edition" of Java (typically used for Applets and desktop applications) with the much simpler "Micro Edition", which is increasingly used on mobile devices such as cell phones and PDAs. After introducing the major features of J2ME, the paper will evaluate the extent to which J2ME addresses the problems identified by the JTF, and look at what needs be done to alleviate any remaining major obstacles to its deployment as a language for the initial teaching of programming.

2. AN OVERVIEW OF J2ME

Despite the "write once run anywhere" slogan, there are actually four different versions of Java:

- J2EE: The Enterprise Edition, typically used for server-side business applications,

- J2SE: The Standard Edition, used for desktop applications and browser-based Applets

J2EE and J2SE are fundamentally differentiated only by the set of APIs assumed to be available. In addition, there is:

- JavaCard: used for smart card applications. This is, in practice, a significantly different language and a basically disjoint set of APIs.

- J2ME: The Micro Edition, used to program applications for mobile/wireless devices. Both a slightly different language and a (largely) disjoint set of APIs.

J2ME [7] is the version of the Java language (and the set of APIs) designed to support applications running on "small devices". As might be imagined, the definition of "small device" is changing quite rapidly as hardware capabilities in mass-market devices increase, but the fundamental constraint on J2ME programs remains the size of packaged (executable) applications – the larger the packaged application is, the more time it takes to download Over The Air, the more time it takes to load, and the more persistent storage it requires on the target device. Significant differentiation in the mobile device market (with/without a camera, with/without BlueTooth, etc.) also has an impact on the APIs which are appropriate for inclusion in particular J2ME systems.

Due to intrinsic hardware restrictions, early versions of J2ME lacked floating point support and other fundamental Java language features, but the increasing capabilities of mobile devices has rendered such restrictions pointless. The remaining significant restrictions to the subset of Java language features available in J2ME are largely related to class-file size: reflection and dynamic class-loading are not supported. In addition recent (Java 5) features such as generics, autoboxing, and enumerated types are not currently supported in standard J2ME (although, given that they have no impact on the Java Virtual Machine which runs Java programs, there is no reason why they couldn't be supported).

Given the wide variation in wireless device capabilities, there is clearly a problem in matching the set of capabilities assumed to be present by application authors against those actually present on a particular device. The J2ME architects have addressed this issue by placing each non-core capability into a separate API package. So BlueTooth, Camera access, Multimedia messaging, etc. are each addressed by separate, optional, API packages. This is in distinct contrast to the approach taken in J2SE, where there is a monotonically increasing set of "core" APIs (e.g. CORBA, XML and AWT/Swing) which are assumed to be present in all environments.

3. J2ME EVALUATED AGAINST THE JTF PROBLEM TAXONOMY

In this section, I discuss how J2ME measures up against the set of problems identified by the JTF, and particularly how it compares in that respect with J2SE. The references in (brackets) are to the problem numbers in Chapter 3 of the JTF rationale document (op. cit.).

3.1 THE GOOD NEWS

In a number of ways, J2ME compares well with J2SE, and addresses a number of problems identified by the JTF: Scale, the use of `static` methods (especially `main`), the lack of a simple input mechanism, the conceptual difficulty of the Graphics model, the appropriateness of the available GUI components, and the inadequate support for event-driven programming.

3.1.1 SCALE (H1)

It is apparent from context that the JTF mean here, and assert that their sources for this problem mean, the scale (extent, scope) of the J2SE APIs, rather than the language itself. This may have been a function of the time at which the JTF was gathering its evidence, as the subsequent release of J2SE 5 (previously known as J2SE 1.5) introduced significant changes to the Java language, as well as to the standard APIs. Some of the language issues the JTF identified are excluded here for the sake of brevity.

The "standard" APIs

It has been observed (by Eric Roberts) that the original text of the "Pascal User Manual and Report" [8] contained some 35000 words. This was enough to specify the entire language, and to provide what was felt to be (at the time) sufficient background material to allow students to learn to program in it. In contrast, there are more than 35000 public methods in the standard J2SE APIs, a figure which increases with every new release of J2SE.

This increase is due not only to the increasing complexity of the "core" (`java.lang`) APIs, but also to the practice of including in the standard distribution of J2SE an increasing number of notionally "optional" packages (`javax.*`). These now includes not only GUI libraries (`javax.swing`), but also XML handling (`javax.xml.*`) and the OMG/CORBA distribution framework (`org.omg.*`). Although these packages are of immense value to professional software developers, the fact that their documentation is integrated into the "standard" online API documentation significantly increases the complexity of a resource which is fundamental to students' understanding of the environment in which they are working. This observation is at the root of the JTF's efforts to exclude "optional" packages from the API documentation visible to students.

In the core (`java.lang.*`) packages, recent releases of J2SE have seen not only the addition of new packages (`java.nio`, `java.util.concurrent.*`) but also significant changes to existing classes such as the use of regular expressions (`java.util.regex`) in the `java.lang.String` class.

In contrast with the approach adopted in J2SE, the J2ME architects have decided to keep the core APIs (assumed to be available in all environments) simple, while providing extra functionality through, separately documented, "optional packages", such as Multi-media messaging and BlueTooth. Merely the ability to not have these packages cluttering up the core (student-visible) documentation provides a significant respite from the information overload observed in programmers first trying to find a particular class in the J2SE documentation.

The Java language itself

After having been stable for a number of releases, the core language defined for J2SE has recently undergone a number of significant changes. These include additions which have long been on the wish-lists of educators (JTF, op cit), most obviously generics and autoboxing [3]. Since the release of J2SE 5, there has not been sufficient pedagogic use of these new facilities to evaluate their effect on practice. They are not yet available in any standards-conformant J2ME implementation although, as they are compile-time only constructs, there is no reason why they could not be made available. See "Stability" below.

3.1.2 STATIC METHODS, INCLUDING main

One of the first problems which must be addressed when teaching application programming using J2SE is the meaning and import of the magic phrase `public static void main (String [] args)`. The necessary appearance of this phrase in students' first program causes two classes of problems: that it refers to a number of language features which will not be introduced until much later in any sensible first course, and that it requires a student's first program to start-up outside the "world of objects". It is this latter issue which seems to cause the most problems.

Students (who are rapidly learning to "trust me and type this") appear to be less concerned by having to type the magic phrase than they probably should be, but are truly confused by the non-availability of the non-static fields and methods they declare as their program's execution progresses.

Courses where J2SE is used to implement Applets do not suffer from this problem (although the magic phrase `extends Applet` is then apparent). J2ME behaves similarly to J2SE-applets in this regard, except that `javax.microedition.midlet.MIDlet` replaces the more familiar `java.applet.Applet` as the controlling class for the application. The MIDlet lifecycle is not radically different to that of the Applet.

3.1.3 LACK OF A SIMPLE INPUT MECHANISM (A1)

Textual I/O (particularly Input) is often the first area of the Java language in which students are exposed to the tensions between Java's commercial and pedagogic uses.

In typical commercial applications, textual output is a marginal concern, restricted perhaps to error logs, and input is handled through complex forms packages or GUI dialogs, not command line prompt-and-respond. In commercial uses of simple textual I/O, failure is a primary concern, and input validation/output formatting are significant issues.

Conversely, in traditional early student programs, (e.g. the classic Fahrenheit-to-Celsius temperature converter) I/O is the major concern of the program, and the application logic which intervenes between I and O is minimal. I/O failure is typically ignored, and input validation/output formatting (if required at all) must be extremely simply implemented.

In the type of applications implemented in J2ME, devices are assumed to have fairly minimal text input capabilities (e.g. predictive text input using a numeric keypad), and the standard APIs provide simple mechanisms for both accessing these facilities and giving hints about the type of input which is expected (e.g. email address, URL, phone number, general text). In practice, these mechanisms and constraints (when suitably incorporated into early assessments) can devolve many of the problems encountered by beginning programmers to the J2ME infrastructure.

3.1.4 CONCEPTUAL DIFFICULTY OF THE GRAPHICS MODEL (A2)

The fundamental issue with the J2SE graphics model, as identified by the JTF, is the necessity for the application to retain enough of its own state to be able to respond to callbacks on its `paint()` method. This is also the model used in the J2ME graphics environment, but the problem is ameliorated by the much smaller (and single) set of available graphical primitives – J2ME does not suffer from the historical duplication of functionality that J2SE exhibits in its `java.awt.Graphics` and `Graphics2D` classes.

3.1.5 GUI COMPONENTS INAPPROPRIATE FOR BEGINNERS (A3)

The J2SE GUI packages (AWT or Swing) contain a large number of disparate components, often with fairly primitive behaviour – there is, for instance, no simple graphical component for typing in a checked numeric value. It is also difficult to build GUIs of the sort of quality even beginning students are familiar with from the simpler of J2SE's provided Layout Managers, leading to frustration for students.

J2ME assumes limited input mechanisms and minimal screen real estate on its target devices. This leads to a number of choices both in the types of GUI component offered, and the flexibility in layout allowed by the standard J2ME GUI toolkit. For instance, all input fields are tagged with the type of input they are to expect. In principle this is to allow appropriate forms of Predictive Text Input to be activated on individual fields, in practice it imposes some quite useful constraints and checks on the data which can easily be input.

Similarly, given the small amount of screen space available, and the primitive nature of item-to-item and screen-to-screen navigation supported on small devices, the flexibility allowed in laying out GUI components in J2ME is also limited, with the effect of vastly reducing the amount of information which students must assimilate before being able to approach the core of the task they are addressing.

3.1.6 INADEQUATE SUPPORT FOR EVENT-DRIVE CODE (A4)

Although, again, J2ME adopts the same style of event-handling as J2SE, the significantly smaller number of different types of event does slightly simplify this aspect of programming. This is aided by the necessity in J2ME of having a single listener for all commands on a particular screen rather than the choice offered in J2SE between a single listener with an internal selection mechanism and per-target listeners. Although the multiple listener style is preferred by OO purists, many libraries designed for beginners, such as ObjectDraw [5], adopt a style similar to J2ME.

3.2 The Bad News

Undoubtedly, there are a number of areas in which J2ME is less suitable as an initial teaching language than J2SE. Primarily, these are: language and API instability, and the availability of tools and textbooks

3.2.1 INSTABILITY (H2)

There is a significant difference between the J2SE approach to adding APIs to the language and that adopted by J2ME (as has been commented above). In some senses, the J2ME approach of adding new APIs in separate, optional, packages has less of an impact on the beginning programmer than the J2SE approach of bundling more and more `javax` packages into the standard distribution. On the other hand, the plethora of optional packages in J2ME makes ensuring that a particular application will run on a particular device some sort of combinatorial nightmare. It also means that individual optional packages can evolve independently, raising the inevitable problem of version incompatibilities.

Another contributor to instability in J2ME is the rapidly increasing capabilities of commercially available target devices. Current generation cellphones are capable of running J2SE language applications, if not supporting the entire array of J2SE APIs. A desire for convergence of development environments and code-bases is generating a pressure for the core J2ME and J2SE language dialects and APIs to merge. In practice, seeing that the major new features of J2SE 5 are visible only to the compiler (i.e. are not visible in the generated Java Byte Code), this issue resolves to that of re-coding the APIs to take advantage of the new language features: a much longer-term problem.

3.2.2 Tools and Texts (H4)

Perhaps the major impediment to the adoption of J2ME as an initial teaching language is the availability of suitable pedagogic tools and text books for beginners to programming.

Many professional IDEs (e.g. NetBeans [10], Eclipse [6], and JBuilder [4]) have facilities (usually optional) for developing, testing, debugging and deploying J2ME applications, and for simulating the target environment. However, these IDEs are notoriously hostile to beginning programmers even in their basic forms, let alone with the addition of plug-ins designed for professional developers. A number of cellphone manufacturers provide their own development environments, although they are for the most part simply a collection of the (sometimes vendor-specific) libraries supported by particular target devices and a bunch of batch scripts to tie things together. Again, this is acceptable for the professional developer, but intimidating for beginning students.

Sun Microsystems "Wireless Toolkit" [11] offers an interesting approach to the problem of providing access to necessary functionality without implementing a complete IDE. It does not concern itself at all with source code creation, management or editing, but provides a simple interface through which J2ME applications can be compiled, simulated and deployed. In combination with a simple text editor, it provides some of the facilities and simplicity required for teaching beginners to programming, without approaching the support that pedagogically-focused IDEs can provide, for example DrJava [2] and BlueJ [9].

The only text books currently available for J2ME programming are firmly targeted at experienced Java developers moving into J2ME. This is a significant impediment to the adoption of J2ME as a first language, but no more so than it was for J2SE when it first emerged.

4. Where Next?

Fitting students' initial experience of programming to their experience as users of computers is a well known challenge for teachers of newcomers to programming. This is particularly hard in the resource-rich environment of desktop or web-based applications, where the sorts of applications students see around them as users are well beyond their ability to produce as beginning programmers.

The use of wireless devices, with their constrained resources, as the target for students' first programs has a number of advantages:

- A large number of students have access to such wireless devices (typically cellphones), but may be new to the realisation that they are programmable devices, still less that they are in a position to program them.

- Given the restrictions on GUI complexity for wireless devices, students can soon come to produce programs

similar in look and feel, to the sorts of applications they use "in real life".

As well as these motivational aspects, the J2ME environment is significantly simpler and smaller than J2SE, but has the massive advantage over all other alternatives in that it is still Java.

Existing CS1 courses which take an "objects first" and "graphics early" approach (whether via a simplified graphics toolkit like ObjectDraw or through "code provided" as suggested by the BlueJ authors) should find that their order-of-presentation of material requires little change and that, although their graphical applications need to be translated to use the J2ME approach, the underlying principles they are trying to communicate are much more simply illustratable in J2ME than in J2SE.

5. REFERENCES

[1] *ACM Java Task Force: Project Rationale.* http://www.acm.org/education/jtf/. Accessed 16 January 2006.

[2] Allen, E., Cartwright, R. and Stoler, B. (2002): DrJava: A lightweight pedagogic environment for Java, in *Proceedings of the 33rd SIGCSE technical symposium on Computer science education.* Washington DC, ACM Press.

[3] Austin, C *(2004): J2SE 5.0 in a Nutshell,* Sun Microsystems. http://java.sun.com/developer/technicalArticles/releases/j2se 15/. Accessed 17 August 2005.

[4] Borland JBuilder, Borland, http://www.borland.com/jbuilder/. Accessed 16 january 2006.

[5] Bruce, Kim B (2001): Bruce, K. B., Danyluk, A., and Murtagh, T. A library to support a graphics-based object-first approach to CS 1. In *SIGCSE Bulletin*. 33, 1 (Mar. 2001), 6-10

[6] Eclipse.org, http://www.eclipse.org. Accessed 16 January 2006.

[7] *Java 2 Platform, Micro Edition (J2ME),* Sun Microsystems. http://java.sun.com/j2me/index.jsp. Accessed 16 January 2006.

[8] Jensen, K. and Wirth, N. (1974): *Pascal User Manual and Report*. New York, Springer Verlag, New York, NY, 1974.

[9] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., (2003): The BlueJ system and its pedagogy. In *Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology*, Vol 13, No 4 (Dec 2003), Swets and Zeitlinger, Rotterdam.

[10] Netbeans.org, http://www.netbeans.org. Accessed 16 January 2006.

[11] *Sun Java Wireless Toolkit,* Sun Microsystems, http://java.sun.com/products/sjwtoolkit/. Accessed 16 January 2006.