



Java for Multi-platform Network Applications

This document aims to describe the benefits and disadvantages of the Java programming language.

Running similar software applications on a wide variety of hardware profiles from a wide choice of manufacturers and software vendors, is attractive to any System Manager, and this has been the goal of Sun Microsystems since they first invented Java in the mid 1990's. Fewer commands need to be learnt and environment maintenance is standardised so that variable definitions (such as '**JAVA_HOME**') and library locations (called '**classpath**' in Java-speak) are easier to manage, across systems small and large. From the developer's viewpoint 'write once - run anywhere' broadens market opportunities, which overall makes software cheaper and more available.

Java programmes written in the Java programming language are 'compiled' into byte-code 'classes' which cause the Java Virtual Machine for the particular device to operate in the desired manner. The original Java Programming language borrowed features from other languages and aimed to improve on them. For example - programmes written in the 'C' programming language - still probably the 'native' or platform-specific language of choice, and which Java closely resembles, require the application developer to do a lot of manual memory-management with the 'malloc' and 'free' commands, within the programme code. This does not make for portability or efficiency. Also Java is inherently 'Object-Oriented' following other languages such as Smalltalk and Delphi. C++ ('C plus plus') is the version of 'C' with Object Oriented capability.

The specification for the Java Virtual Machine which runs the Java classes on a device includes a mechanism called a 'garbage-collector'. This periodically checks the system during the running of the programme, detects unused objects, and deletes them thereby recovering memory. Alternatively it may look at the System resources and the needs of the programme and allocate more memory to the programme. This is more efficient than traditional methods because the JVM knows more about the system it is running on in that instant than the programmer who may have written the code years before.

Object Orientation is an approach to analysing the parts of a System in terms of characteristics, like 'size' or 'colour', and activities, such as 'print', or 'run'. Java enables us to use the same families of object classes as a common basis and optimise system design. These classes are arranged in a hierarchical manner and features are inherited. For example, in human terms, the activity 'paint' can be applied to say a house, or a car - they are both in the family of objects which can be painted, even though the detail of the activities are vastly different. In our Object Oriented programme, at the higher level we just specify the command 'paint(object)' and the system knows the type-family of our object and applies the appropriate (sub) method for the house or car accordingly. This means that we do not have to re-write the code at the higher level to include say, a motor-home in our paint programme, and we can arrange for the motor-home object class to borrow from methods we might have written for the house or for the car, as appropriate. In practice, the benefits for Software development and maintenance are huge.

Sun Microsystems licence Java products and the specification process has always been open and largely democratic within the Java Community, and Sun has now released the source code for their products. A lot has been learnt over the 5 major releases of the Java Standard Edition. We are currently at J2SE 1.5 and Java has been called J2 since version 1.2. Originally developers had no control over memory management - it was foreseen this would not be needed. With subsequent releases however, not only can memory limits be applied to the JVM at start-up to make programme execution more efficient, but the 'garbage-collector' can now

be 'advised' when to run, within the programme. Java Objects are strongly 'type-checked' by the 'compiler' and Java Virtual Machine. Furthermore, unlike objects written in say C++ Java objects are aware of their individual characteristics and capabilities. This self-awareness means that 'reflective' methods may be used to introspectively examine the state of the programme, which means code can be more general-purpose.

The Java language has good built-in error catching and handling mechanisms. Access control is also formalised, with 'private' and 'protected' objects and methods (procedures) which can only be accessed within the same class or package respectively. At a higher level, 'Applets' are Java programmes which can only be run within a web-browser or Applet-Viewer. Ordinarily applets are not allowed to access the local file-system and network access is restricted. Midlets - written for hand-held devices are similarly protected from the platform.

Java classes are packaged in Java archive or 'jar' files. These are compressed by default to save on network bandwidth and storage space. A manifest file gives the Java engine meta-information and the Jar file can be digitally signed to over-ride default network and storage permissions.

Netscape released 'Javascript' about the same time as Java was released and unfortunately is probably used more for Web client programming than Java. Many Java code errors are found at compile-time whereas efficient Javascript relies more on testing with all known interpreters. Around the time of Netscape 4.x and Internet Explorer 4.x it was often the case that more Javascript code was dedicated to identifying the platform and browser than performing the actual processing and those who strayed from the popular platform and browser combination often experienced problems. There are at least 3 international standards for Javascript type languages and numerous implementations. Javascript has many similarities with Java in that it is also Object-Oriented and the code also resembles 'C', but sometimes Javascript is better integrated with the browser. In this case the mutual access between Java and Javascript objects provided by many browsers, including Internet Explorer using Netscape classes, is useful.

There are 3 Java editions - Java 2 Standard Edition (J2SE or Java SE) is used for a wide range of Applications and Applets for PC type platforms, but assumes significant processing power, bandwidth and memory, compared with Java 2 Micro Edition (J2ME or Java ME) applications for memory, power and size constrained hand-held devices. Java 2 Extended Edition (J2EE) takes a modular approach to code production with plug-in Application Programming Interfaces which are normally used by large system developers such as web-service providers.

Most web clients are still relatively 'Thin' - that is most processing is done at the Server with high powered devices. This is changing due to the greater availability of storage, memory and improved bandwidth. However there is also a demand for services for mobile devices without these luxuries. With careful design, notably avoiding J2EE, we can achieve a high degree of flexibility of architecture between Client and Server - even dynamic selection of Thin or Rich Client services depending on resources.

Within Java 'widgets' - components used in data entry forms such as Buttons, TextFields and TextAreas; Choice Lists, and Checkboxes - were originally handled by the 'Abstract Windowing Toolkit' (AWT). Around 1998 Java was perceived as being slow, so a new Toolkit - 'Swing' addressed this providing improved 'Look and Feel' choices and responses assuming the greater availability of memory. The 'Hotspot' compiler is also more intelligent and flexible than the classic compiler. The byte-code 'compiled' from Java source-code which is interpreted and run by the JVM is now likely to be compiled to native code by the 'Just In Time' compiler. This is the reason a Java application will speed up once launched. Since 2003, just as the industry caught up with Swing, releases of AWT have proved measurable improvements. The mobile device industry learnt a lot from the Web industry and minimum Java device standards CLDC1.0 and MIDP1.0 are secure but more flexible than Java Applets for Web devices.

Java applications are inherently inefficient. The intermediate stage of writing for a Java Virtual Machine which then processes the byte-code is not as efficient as writing directly for a particular platform, and despite improvements the Java Virtual Machine is still a general purpose device which must make assumptions about its environment. The Gnu Java Compiler (gcj) from the Free Software Foundation compiles and links Java classes to produce native code or executables. This adds a new dimension to Applications Development using Java, and could well have an impact even on the development strategies of high performance and specialised products. Within a development strategy Java is therefore the language of choice.

Most consumers prefer to use products which adhere to professional or industry standards because information on performance and other parameters is available; there is normally a greater choice of products, and system compatibility problems are reduced. Over the decades in which Internet type networking has been developing, administrative, professional and industrial bodies have become more skilled in defining accurate specifications which allow for flexibility.

The Java Programming Language allows the same code to be run on high powered servers, desktop PC's and Mobile Phones and other devices. Sun Microsystems licence Java products and the specification process has always been open and largely democratic within the Java Community, and Sun has now released the source code for their products. A lot has been learnt over the 5 major releases of the Java Standard Edition. Most major Mobile device manufacturers, among others, were involved in defining the J2ME language specifications for Connected and Connection Limited devices.

Information transferred over the Web and similar networks is increasingly defined in XML (Extensible Markup Language) - standards defined by the World Wide Web Consortium. These specifications help to provide flexibility, ease of use and control of Information Content, Structure and Presentation. See how TelForms uses XML data definitions to automatically generate java Web Forms for a range of platforms.

See <http://www.terry-comms.com> or <http://www.telform.info> for Mobile devices.

Copyright © terry-comms 2003-2010 version-20100817 : 1703 |