

# **Portability Analysis of Java Source Code**

THESIS

Submitted to the Graduate Council of

Southwest Texas State University

in Partial Fulfillment of

the Requirements

For the Degree

**Master of SCIENCE**

By

**Tanvir Rahman, B.E.E.E.**

San Marcos, Texas

August 2000

**To**  
**Leema**

# Acknowledgements

All praise is due to Allah.

I would like to thank Dr. Greg Hall for his valuable guidance and advice during the entire thesis work. I also would like to thank my dear wife, Leema for her extreme patience, encouragement and support during the research work.

Tanvir Rahman

Southwest Texas State University

August 2000

## **TABLE OF CONTENTS**

### **CHAPTER 1**

1.1	INTRODUCTION.....	1
1.2	ELABORATION OF THESIS STATEMENT.....	2
1.3	DEFINE "PORTABILITY".....	3
1.4	JAVA AND PORTABILITY .....	4
1.5	THE IMPORTANCE OF PORTABLE SOFTWARE.....	8
1.6	JAVA'S PORTABILITY FEATURE IN BUSINESS APPLICATION.....	9
1.7	SOFTWARE PORTABILITY ISSUES.....	11
1.8	MEASURING SOFTWARE PORTABILITY.....	14

### **CHAPTER 2**

2.1	JAVA'S ARCHITECTURE FROM PLATFORM INDEPENDENCE PERSPECTIVE.....	16
2.2	JAVA PROGRAMMING LANGUAGE.....	17
2.3	JAVA CLASS FILE AND API.....	18
2.4	JAVA VIRTUAL MACHINE.....	20
2.5	ALTERNATE APPROACH OF THE JAVA VIRTUAL MACHINE.....	21
2.6	PORTABILITY AT THE COST OF EFFICIENCY.....	23



2.7	RESEARCH PROJECTS.....	25
<b>CHAPTER 3</b>		
3.1	INTRODUCTION OF RESEARCH METHODOLOGY.....	28
3.2	DESCRIPTION OF RESEARCH METHODOLOGY.....	31
3.3	ANALYSIS OF PRELIMINARY TEST DATA .....	32
3.4	FINAL TEST METHODS.....	34
3.5	JAVA TEST CODE.....	36
3.6	TYPES OF SYSTEMS, OPERATING SYSTEMS AND VERSIONS OF JDK / JRE USED.....	37
3.7	TEST RESULT OVERVIEW.....	38
3.8	ANALYSIS OF TEST RESULT.....	44
<b>CHAPTER 4</b>		
4.1	INFLEUNCING FACTORS ON JAVA'S PLATFORM INDEPENDENCE.....	61
4.2	RESEARCH LIMITATIONS.....	66
4.3	FUTURE RESEARCH.....	66
4.4	CONCLUSION.....	66
<b>APPENDIX A.....</b>		70
<b>APPENDIX B.....</b>		93
<b>APPENDIX C .....</b>		128
<b>WORKS CITED.....</b>		129
<b>VITA.....</b>		131

## LIST OF TABLES

Table 1. Java data types.....	30
Table 2. Result abbreviation.....	39
Table 3. Result abbreviation.....	70
Table 4. Result overview of test 1.....	71
Table 5. Test configuration 1- Compiled on Intel system under Win95 OS.....	75
Table 6. Test configuration 2: Compiled on Intel system under Win98 OS.....	78
Table 7. Test configuration 3 - Compiled on Intel system under WinNT OS.....	81
Table 8. Test Configuration 4: Compiled on Apple system under Mac OS.....	84
Table 9. Test Configuration 5: Compiled on Intel system under Linux OS.....	87
Table 10. Test configuration 7: Compiled on Sun Workstation under Solaris OS.....	90
Table 11. List of vendors that support Java portal work.....	128

## LIST OF FIGURES

Figure 1. Java on different platforms.....	16
Figure 2. Diagram of Java virtual machine.....	21
Figure 3. First method: compile .java files, and run .class files on all platforms.....	34
Figure 4. Second method: compile .java file on a platform, and run the outputted .class files on all platforms.....	35
Figure 5. Compilation and execution results overview.....	39
Figure 6. Execution results of the test programs that were compiled under Win95 platform.....	40
Figure 7. Execution results of the test programs that were compiled under Win98 platform.....	41
Figure 8. Execution results of the test programs that were compiled under WinNT platform.....	42
Figure 9. Execution results of the test programs that were compiled under Macintosh platform.....	42
Figure 10. Execution results of the test programs that were compiled under Linux platform.....	43
Figure 11. Execution results of the test programs that were compiled under Solaris platform.....	43

Figure 12. Snapshot of Hello2.java under Win95	
Platform (correct output).....	48
Figure 13. Snapshot of Hello2.java under Macintosh	
platform (wrong output).....	48
Figure 14. Snapshot of Hello2.java under Linux platform	
(wrong output).....	49
Figure 15. Snapshot of FrameSize.java program under Linux	
platform (correct output).....	57
Figure 16. Snapshot of FrameSize.java program under Win95	
platform (wrong output).....	58
Figure 17. Snapshot of GetSourceBug.java program under Win95	
(left - correct output) and Win98 (right - wrong	
output) platforms.....	59
Figure 18. Snapshot of labelbug.java program under Win95	
(left - wrong output) and Linux (right - right	
output) platforms.....	60
Figure 19. Snapshot of EmptyMenubarBug.java program under	
Win95 platform (wrong output).....	94
Figure 20. Snapshot of EmptyMenubarBug.java program under	
Linux platform (correct output).....	95
Figure 21. Snapshot of EmptyMenubarBug.java program under	
Macintosh platform (wrong output).....	96

Figure 22. Snapshot of TrivialApplication.java program under	
Macintosh platform (wrong output).....	99
Figure 23. Snapshot of HelpMenu.java program under Linux	
Platform (correct output).....	102
Figure 24. Snapshot of HelpMenu.java program under Win95	
platform (wrong output).....	102
Figure 25. Snapshot of SetSizeBug.java program under	
Win95 platform (wrong output).....	108
Figure 26. Snapshot of SetSizeBug.java program under	
Linux platform (correct output).....	108
Figure 27. Snapshot of test2.java program under Win95	
platform (wrong output).....	109
Figure 28. Snapshot of test2.java program under Macintosh	
platform (correct output).....	110
Figure 29. Snapshot of TestCase.java program under Win95	
platform (wrong output) .....	111
Figure 30. Snapshot of TestCase.java program under	
Macintosh platform (correct output).....	112
Figure 31. Snapshot of testPopup.java program under Win95	
(left - wrong output) and Linux	
(right - correct output) platforms.....	114
Figure 32. Snapshot of WindowTest.java program under	
Win95 platform (wrong output).....	120

Figure 33. Snapshot of Factors42.java program under Win95

platform (wrong output).....122

Figure 34. Snapshot of ColorTest.java program under Win95

platform (wrong output).....124

# CHAPTER 1

## 1.1 INTRODUCTION

One of the biggest challenges presented to software developers by today's network-centric hardware environment is the wide range of devices that are connected to a network. A typical network usually has many different kinds of attached devices having diverse hardware architectures and operating systems. Java addresses this challenge by providing platform-independent programs. Sun Microsystems, the originator of the Java language, claims that a single Java program can run, unchanged, on a wide range of computers and devices. According to Sun, the "write once run anywhere" (WORA) capability is inherent to Java architecture. It is this WORA capability that brings the burgeoning popularity of Java in today's information technology (IT) industry. According to Hudgins-Bonafield,

Java development already is mainstream in the top 20% to 30% of IT organizations and is on the verge of becoming mainstream in the rest, says Gartner Group Inc. analyst David Smith. Similarly, Forrester Research Inc. finds that almost half of Fortune 1,000 companies already use Java--and that nearly 20% consider it important or critical. By year 2000, Forrester predicts, nearly half of the Fortune 1,000 will consider Java important or critical, with 80% of those companies relying on it as their dominant application development language. (1998)

Many of the organizations are writing or are considering writing applications in Java to meet their business needs. What they do not know is how well their applications will run on different platforms. For example, how well

does an application written on NT run on a Macintosh? The main purpose of this research is to find the answer to this question.

## **1.2 ELABORATION OF THESIS STATEMENT**

The notion of software portability is not new. Platform independent code can be written using the C or C++ language by strictly following a portability guideline. But Java adds a new dimension to the definition of portability. A Java program does not require recompilation to run on different platforms. According to Sun Microsystems, it is possible to write complex multithreaded Java applications with complex user interfaces that run immediately and without recompilation or modification on all platforms. The general consensus is the Java language is platform independent by nature, and no portability standard needs to be adhered to achieve this goal.

Although there is great enthusiasm about Sun's WORA claim regarding Java, in reality, there has been little research performed, outside of Sun Microsystems, to support the claim. Therefore, this research will play an important role in validating the WORA capability of Java. The main goal of this research is to gather and analyze empirical data to find out how well Java lives up to its promise in offering WORA capability. The questions this research attempts to answer are:

- Does Java version 1.1.8 provide full WORA capability?



- Can code written to a particular Java language standard be portable (without following additional development guidelines)?
- Does Java version 1.1.8 provide graphical user interface (GUI) portability?

To answer the above questions, a wide variety of Java source code is compiled and run on different system configurations. The source codes are written such that they address a wide range of generic portability issues.

Six different operating systems (Win95, Win98, WinNT, Macintosh, Solaris, Linux) running on four different hardware platforms (Pentium processor based systems of Hewlett Packard and IBM, AMD K-6 processor based system of Dell Computers, G3 Macintosh system of Apple Computers, and E450 Sparc system of Sun Microsystems ) are used to test the source codes.

### **1.3 DEFINE "PORTABILITY"**

"Portability" is defined (Warr 1974) as the ability to economically move a program from one computer to another. "Economically" means that it is an order of magnitude cheaper to move the program than to rewrite the entire program. According to *The Prentice-Hall Standard Glossary of Computer Terminology* by Robert A. Edmunds, portability is defined as follows:

Portability: A term related to compatibility. Portability determines the degree to which a program or other software can be moved from one computer system to another. (1984)

Portability can be discussed from two points of view: generic and specific. Generically, portability simply means running a program in a host environment that is somehow different from the one for which it was designed. Since the cost of producing and maintaining software far outweighs the production cost of hardware, it is desirable to reuse the same software in newer hardware models.

Specific portability involves identifying the individual target environments in which a given program must run and clearly stating how the environments differ. This research concentrates on finding the generic portability issues of the Java language.

#### **1.4 JAVA AND PORTABILITY**

From the above definitions we can conclude that portability refers to the ability to run a program on different machines. Running a given program on different machines can require different amounts of work. For example, it may require recompilation, or making small changes to the source code, or no work at all. When people refer to Java applications and applets as portable, they usually mean the applications and applets run on different types of machines without any change. This notion came from Sun Microsystems' claim of Java's WORA capability.

Java technology brings with it three distinct types of portability - source code portability, central processing unit (CPU) portability, and operating system/graphical user interface (OS/GUI) portability. Each is independent of the

others, but the combination of the three provides Java with much of its power and promise.

### ***Java as a programming language - Source code portability***

A given Java program should produce identical results regardless of the underlying hardware, operating system, and Java compiler. This idea is not new; languages such as C and C++ have provided the opportunity for this level of portability for many years. But, unless programs written in C and C++ are designed to be portable from the beginning, the ability to move them from one machine to another is more theoretical than practical. This is mostly because of the semantic variations of C and C++ programs. The semantic looseness allows a single block of C or C++ source code to compile to programs that give different results when run on different processors, operating systems, and compilers. Sun claims that Java is different from this since it provides much more rigorous semantics and leaves less up to the programmer.

Additionally, Java defines more behaviors than C and C++. Different features of Java that are discussed later in this thesis also help to understand how Java narrows down the variation in the behavior of a program from platform to platform and implementation to implementation. Even without the Java virtual Machine (JVM), programs written in the Java language are expected to port (after recompiling) to different processors and operating systems much better than equivalent C or C++ programs.

### ***Java as a virtual machine - CPU portability:***

Most compilers produce object code that runs on one family of processors (for example, the Intel x86 family). Even compilers that produce object code for several different processor families (for example, x86, MIPS, SPARC), produce object code for one specific processor type at a time. The source code has to be recompiled multiple times, each time producing object code for a specific processor family.

Again, Sun claims that Java does not have this drawback. Instead of producing output for a specific processor family, Java compilers produce an intermediate code called byte code. For each real processor the Java virtual machine executes the byte code. Thus the Java virtual machine allows the same byte code to run on any processor for which a Java interpreter exists. The JVM, being essentially an imaginary processor, is independent of the source code language.

### ***Java as a virtual operating system - OS/GUI portability:***

Most Microsoft Windows programs written in C or C++ do not port easily to the Macintosh or Unix environments, even after a recompilation. Even when the programmers take extra care to deal with the semantic weaknesses in C or C++, the port is difficult. This difficulty arises as windows programs make various calls to the operating system that are very different from those of Macintosh and Unix programs. These native calls are critical in writing non-trivial windows programs.

Java solves this problem by providing a set of libraries that talk to a virtual OS and a virtual GUI. Just as the JVM presents a virtual processor, the Java libraries present a virtual OS/GUI. Java provides a least-common-denominator functionality in its OS/GUI libraries. The advantage to this approach is that mapping the common functionality to the native OS/GUI is fairly easy and, with care, can provide applications that work as expected on most operating systems.

In *100% Pure Java Cookbook for Developers – Rules and Hints for Maximizing Portability*, Sun Microsystems states that there is a common misconception about Java's portability in people's minds. People like to believe that a given Java program should produce identical results regardless of underlying hardware and operating systems. But in reality, a graphical user interface based program written in Java shouldn't generate "identical results" on different platforms. According to Sun, the platforms may exhibit cosmetic user interface and "look and feel" differences, but as long as a program generates the common behavior, it is a portable program. Common behavior does not mean identical behavior, but functional behavior conformant with the underlying platform. Sun Microsystems thus defines a portable program as "one that fulfills its function on any platform." (Sun Microsystems Press, 1999)

## 1.5 THE IMPORTANCE OF PORTABLE SOFTWARE

The importance of portable software is manifold.

- According to Warren, J.C, portable software appeals to a wider market niche. It is economic in the long run since it reduces the effort to generate multiple versions of the same software for various platforms and environments. Besides, the overall cost of developing a portable program once is bound to be advantageous compared with rewriting the same program for different environments several times. (1974)

- Portable software bridges varieties of hardware and system software with a common language, and thus opens up new markets.

- According to Henry Rabinowitz and Chaim Schaap, since generating and maintaining software is more expensive than hardware, its life cycle must be made longer so that it survives multiple hardware evolutions.(1989)

- In today's network-centric world, portable software presents a positive sales argument.

- Portable software brings new freedom to users to choose and change hardware at will.

- Indirectly, by making it easier to switch operating systems, portable software puts pressure on OS vendors to provide better products, and respond to customer feedback.

- Maintaining different types of software that run on different types of systems is always troublesome for information system (IS) departments. Portable software allows them to reduce the number of software they have to maintain.

This section presents the importance of portable software in general. The next section discusses how Java as a portable language is useful in real world applications.

## **1.6 JAVA'S PORTABILITY FEATURE IN BUSINESS APPLICATION**

Robert W. Atherton, in his article, *Moving Java To The Factory* (1998), presents Java's potential in manufacturing facilities. The Java programming language, the runtime environment, and the Java based microprocessors are the main three components of Java technology that provide platform independence and an internet integration facility. The security, portability and embedded application creation facility are some of the major attributes of Java that make it a powerful platform for industrial control and control system development. Java is a portable language that can deploy applications over the World Wide Web. It is object-oriented in nature and its design process avoided most of the shortcomings of other object-oriented languages. For example, it does not allow pointer manipulation, multiple inheritance, or operator overloading. It has no direct access to memory and no extended constructors. On the contrary, it has some very important features such as exception handling which makes it easy to change the flow of a program when some unexpected event occurs. It also has an

automatic garbage collection facility, a strong security model, and built-in features for web access. Java achieved its cross-platform independence through its platform neutral architecture.

Java has many technical advantages over conventional client/server implementation. Because of its platform independent nature, any Java application can run on any Java-enabled computer; thus it reduces the different versions of applications that may otherwise be needed. This is a significant help in industrial automation as many different types of platforms are usually found in industrial environments. Moreover, its security and safety features are two of the major requirements to build industrial applications.

Java technology can also be widely used in embedded control system design. Because embedded systems use a variety of microprocessors, they can be well served by Java technology's platform independent feature. Additionally, Java's garbage collection feature prevents memory leaks that are costly to fix in an embedded device once it is deployed.

The software technology demands of enterprise control systems are very complex and challenging. Currently such an enterprise is controlled through various large software applications along with an enterprise resource planning system. Some of these applications are electronic design document (EDA), computer-aided design and manufacturing (CAD and CAM), product data management (PDM), supply-chain management (SCM), various database applications and so forth. Each of these software applications usually requires a network of desktop computers, application servers, and database servers.



Furthermore, to effectively control the enterprise, these software applications need to interact with each other. The poor performance of today's ERP and SCM systems stem from their inability to access data from process control layers. These problems become more complicated in case of a heterogeneous hardware environment. The author believes that the Java architecture is able to resolve most of these issues. Since Java is network-ready, the development and deployment of distributed applications is very simple using Java. The native database connectivity support is a core feature of the Java environment. Furthermore, Java's cross-platform nature helps in automation among various platforms. Finally, Java's intelligent agents can function at any layer of enterprise control from the manufacturing floor to enterprise planning.

Enterprise-control-applications for large manufacturing companies were a big challenge until the emergence of Java technology. The author believes that Java technology combined with good system engineering is the answer to true enterprise integration and control.

## **1.7 SOFTWARE PORTABILITY ISSUES**

There are some major problems that need to be addressed to produce portable software. In this section, some of the major issues that hinder software portability are discussed from P.J. Brown's book *Software Portability* (1977). In this research, various test codes have been written to explore how Java handles the following portability issues.

- ***Internal representation of data***

One of the governing characteristics of a system is the length of a word. The length varies from 8 to 64 bits. It affects the method of representing values in integers and other numbers, the method of addressing data, the number of characters permissible in a word etc. Besides, the size of internal data such as *char*, *integer*, *floating-point* etc. differs across platforms. Furthermore, the ordering of bytes within words and words within long words also differs on various platforms. Such encoding schemes are referred to as big-endian and little-endian.

- ***Numeric types***

**Integers:** Integers present problems in computational operations, such as division, that do not have integer results. Depending on the machine, such computation can result in truncation or rounding.

**Real numbers:** Real numbers can only be approximately represented, more or less accurately. Therefore, every operation on a real number yields an approximate result. When it is repeated several times, it may cause major errors on different platforms. The result thus depends on the representation of the real number and the precision with which it is obtained.

- ***Character sets***

In environments where the target is not the system on which the software is being developed, differences in character sets becomes an important issue. There are several types of character sets, such as:

- 7 bit: ASCII (American Standard Characters for Information Interchange)
- 8 bit: EBCDIC (Extended Binary Coded Decimal Interchange Code)
- 16 bit: Unicode

Hence, there may be problems with the character set to be used as each manufacturer supports only a single set.

- ***CPU speed***

When an execution of a process or thread depends on CPU speed, it poses a threat to portability. This may result in an unexpected behavior of a given program on different platforms.

- ***Operating system***

The principal issues here are single versus multitasking, and fixed versus virtual memory organization in operating systems. The difference of any of those on different platforms may cause programs to be non-portable. Furthermore, thread synchronization or prioritization is also a major concern in creating portable software. Besides, accessing system resources often requires invoking native methods and that breaks platform independence.

- ***File system:***

File systems are also a major concern in software portability. Whether multiple versions of the same file can coexist, or whether the date and time of creation or last modification are stored is platform dependent. Furthermore, number of characters permitted in file naming, and whether or not such names are case sensitive, are also major concerns in software portability.

## 1.8 MEASURING SOFTWARE PORTABILITY

How would one know whether software portability has been achieved? Is it when the code compiles and links without error? Or, is it when all target machines produced the same output?

Before the emergence of Java, one group of scholars believed that a true sense of portability is achieved when a program is compiled and linked without error. But because of implementation-defined behavior it may be possible to get different results in different target machines. The legitimate results may even be sufficiently different as to render them useless. For example, floating-point range and precision may vary considerably from one target to the next such that results produced by the most limited floating-point environment are not precise enough. The look and feel of the graphical user interface of a portable program may also vary from platform to platform because of the underlying operating systems. Yet another group considered a software product to be portable if the data is ported correctly. In fact, until the introduction of Java, one could not tell whether or not software has achieved true portability without adequately defining the specific portability scenarios.

Java helps in providing a true standard in measuring software portability. It takes the definition of portability a step further. According to Sun Microsystems, recompilation should not be needed for a portable software to run on multiple platforms. Furthermore, the graphical user interface of portable

software may not produce the same look and feel across different platforms but the output and data representation should be identical on all platforms.

## CHAPTER 2

### 2.1 JAVA'S ARCHITECTURE FROM PLATFORM INDEPENDENCE PERSPECTIVE

Bill Venners in his book *Inside The Java Virtual Machine* presents Java architecture from three interrelated layers (1999):

- Java programming language
- Java class file format and application program interface (API)
- Java virtual machine (JVM)

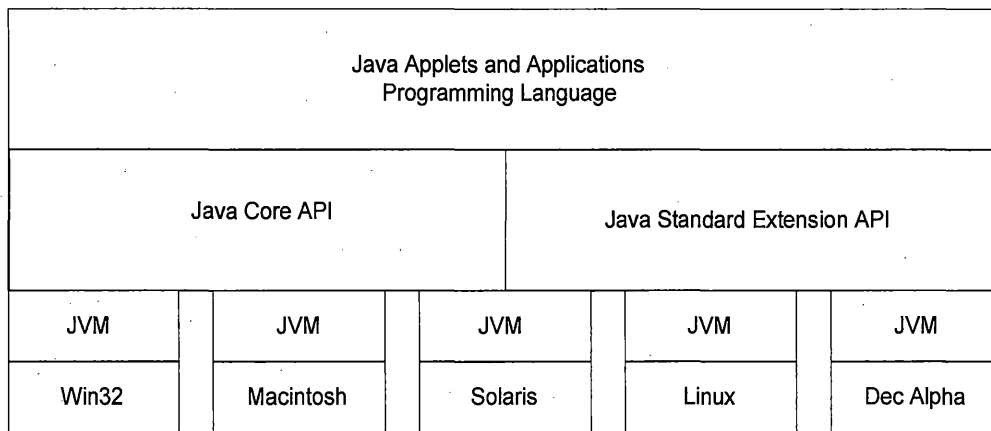


Fig. 1. Java on different platforms.

This section briefly describes Java architecture layers from a portability standpoint. Java's architecture and feature set has been the subject of various research efforts. This chapter reviews some of the technical articles written based

on previous research. In addition, this chapter introduces two Java related projects that are currently in progress.

A Java source file is written using the programming language. The source file is then compiled into class files, and the class files are finally run on a virtual machine. The combination of Java virtual machine and Java API is called the *Java Platform*. Since the Java platform itself is implemented in software, it acts as a buffer between a running Java program and the underlying hardware and operating system. Java programs are compiled to run on a virtual machine, with the assumption that the class files of the Java API will be available at run time. The virtual machine runs the program, while the API gives the program access to the underlying computer resources. Hence no matter where a Java program runs, it only needs to interact with the Java platform. As a result, the application can run on any computer that hosts a Java platform.

## **2.2 JAVA PROGRAMMING LANGUAGE**

In *Java and the Shift to Net-Centric Computing* (1996), Marc A. Hamilton describes various features, strengths, and built-in security mechanisms of Java. Although Java is widely used, the author believes that its potential is still less understood.

Java was originated in early 1990 by James Gosling, a software developer at Sun Microsystems. It is an object-oriented language with a much simpler syntax than C or C++. It has a robust memory management and security scheme

and it supports multithreading. But according to the author, Java's platform independence feature far exceeds the other features and makes it more attractive.

Unlike C, Java data types are independent of the underlying hardware and operating systems. For example, an *integer* is always 32 bit in Java regardless of underlying 32 bit or 64 bit processors. Besides, Java uses a 16 bit Unicode character set. Contrary to C, Java arrays are true objects with length and bounds checks that happen both at compile time and run time. By enforcing strong type checking and eliminating the pointer, Java helps in reducing program errors and security flaws, and thus provides better platform independence.

### **2.3 JAVA CLASS FILE AND API**

Java class files have a binary format that gets translated into byte codes for a virtual machine. Therefore, Java class files can run on any hardware platform and operating system that hosts a Java virtual machine. This breaks the traditional approach of C or C++ programs as programs written in these languages are compiled and linked into binary executable files that are specific to a particular hardware platform and operating system. C, C++ executables contain machine languages that are specific to a target processor.

In addition to processor-specific machine language, another platform-dependent attribute of a traditional binary executable file is the byte order of integers. The byte orders of binary executable files for Intel X86 family processors is little-endian or lower order byte first. The byte order for the



PowerPC chips, on the other hand, is big-endian or higher order byte first. In Java class files, byte order is big-endian regardless of the platform that generated the file and independent of the platform that may eventually use the file.

The Java API is a set of run-time libraries that gives a standard way to access system resources of a host computer. The class files of the Java API are inherently specific to the host platform. To access the native resources of the host, the Java API calls native methods. Thus the top user program does not do this directly. The Java API class files thus provide a standard, platform independent interface to the underlying host.

The inherent design of Java API is also geared towards platform independence. For example, the graphical user interface libraries of Java API, the *Abstract Windows Toolkit* (AWT), and *Swing* are designed to facilitate the creation of user interfaces that work on all platforms. The AWT implementation relies on and uses the underlying native window system. This encourages the adaptation of the look and feel of the underlying platform. The *swing* library, on the other hand, makes no use of native window system semantics. It is a full set of GUI components such as scrollbar, button, menus, etc., that are all written in Java. At the lowest level, *Swing* uses a drawing surface (e.g., a canvas, window etc.) from the native window to render these components. Since all the *Swing* components are written in Java, they work the same way on all platforms.

## 2.4 JAVA VIRTUAL MACHINE

The Java virtual machine is an abstract operating system that supports the three major characteristics of Java language: platform independence, security and network mobility. The flexible nature of its specification enables third party vendors to implement it on a wide variety of computers and devices.

The JVM is a stack machine. The instructions in this machine are encoded in a compact form of variable length, with the shortest instructions occupying 1 byte and most instructions being 1 to 3 bytes long. This form of encoding is known as byte code. The JVM has two main components: a class loader and an execution engine. The main task of a virtual machine is to load class files and execute the byte codes they contain.

The execution engine in the virtual machine is implementation dependent. Currently, there are three different kinds of execution engines:

- The simplest kind of execution engine just interprets the byte codes one at a time.
- The second one is called *Just-In-Time* (JIT) compiler that is faster than the first one but requires more memory. In this scheme, the byte codes of a method are compiled to native machine code the first time the method is invoked. The native machine code is then cached, so the code can be reused the next time when that same method is invoked.
- The third type of execution engine is called *Adaptive optimizer*. In this scheme the virtual machine simultaneously interprets the byte codes, and monitors the activity of the running program and identifies the most heavily used

area of the code. As the program runs, the virtual machine compiles to native machine code and optimizes the heavily used areas. The rest of the code remains as byte code - which the virtual machine continues to interpret.

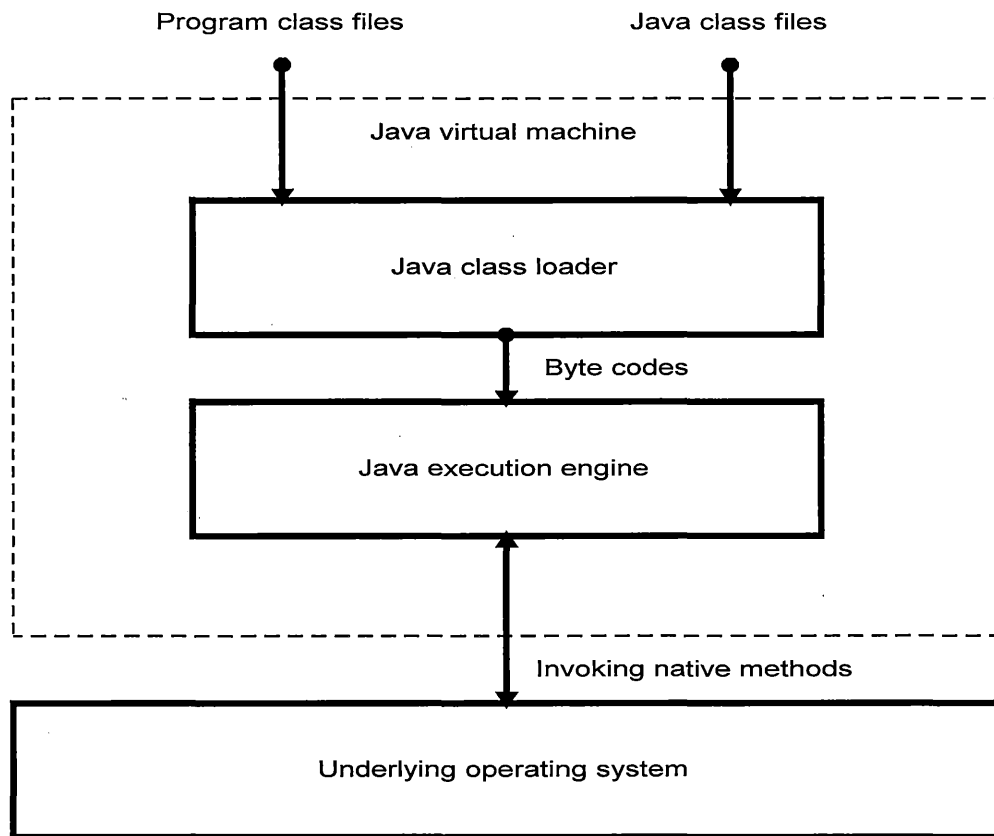


Fig. 2. Diagram of Java virtual machine.

## 2.5 ALTERNATE APPROACH OF THE JAVA VIRTUAL MACHINE

Mike O'Connor in his article, *PicoJava: A Direct Execution Engine For Java Byte code* (1998), presents an alternate solution of the Java virtual machine. Java program runs on a virtual machine that insulates it from any contact with the

underlying hardware. This article introduces a small, flexible byte code execution engine called "PicoJava" that directly executes Java byte code instructions, and provides hardware support for other essential functions of the Java virtual machine.

The primary advantage of byte code is its ability to create a single image of a program that executes identically on any system that has a Java virtual machine. The performance of a Java program depends not only on the speed of the underlying platform, but also on the effectiveness of the dynamic translation technology that converts byte code instructions into native machine code instructions. Java processors were developed to address this performance issue. Java processors are microprocessor devices that execute byte code instructions directly in hardware, bypassing dynamic translation. The intention was to provide the same high performance that sophisticated dynamic compilers provide in a small-footprint device, and thus extend platform independence in embedded environments.

The PicoJava core is the byte code execution engine of Java processors. The major blocks of in this core are the integer execution unit and the compact floating-point unit to support the floating-point specification of the Java virtual machine. The target for this engine is the class file generated by the Java compiler.

By eliminating the need for dynamic translation, picoJava core provides a substantial performance boost for byte code programs. This article presents an efficient way of achieving software portability by replacing the software-based

virtual machine with a hardware device. This approach eliminates the efficiency drawback of the Java programs. The implementation variations of the software based JVM causes a lot of portability concerns. The hardware based JVM presented in this article should eliminate most of the portability issues that stem from different third party vendors' implementations of the JVM.

## **2.6 PORTABILITY AT THE COST OF EFFICIENCY**

Java is known to be a highly portable and safe language, but it lacks efficiency. Several solutions including just-in-time (JIT) and offline byte code compilers have been proposed to overcome this tradeoff, but unfortunately most of the solutions cause Java to lose either its portability feature or its ability to dynamically load byte code. In the article, *Harissa: A Hybrid Approach to Java Execution* (1999), Gilles Muller and Ulrik Pagh Schultz present a hybrid solution that can resolve all these issues.

Harissa is an efficient offline compilation system that fully supports dynamic byte code loading. It consists of an optimizing byte code to C compiler and an interpreter that is integrated into the runtime library. The Harissa compiler replaces stack management with variables and virtual method calls. It generates faster, optimized code for single threaded programs, and produces C code that executes more efficiently than other alternate approaches.

The authors provide some background information on several strategies that have been proposed to optimize Java execution time. One of those strategies

is compiling source code into native code. But since Java programs are distributed in byte code format, it is not possible for the end users to compile such programs for a specific platform. Another strategy is to use the byte code compilers that take byte code as input. But the Java byte code contains the same amount of information as source code. The third approach is the just-in-time compiler that compiles code on an as needed basis at runtime. The drawback in this approach is that optimization becomes an overhead during execution. Furthermore, the availability of the JIT compilers is platform dependent.

There are two types of offline byte code compilers: native and non-native. Native compilers directly produce executable code and hence speed is an advantage of this type of compiler. Non-native compilers, on the contrary, produce code in an intermediate language. These compilers are more flexible and generate portable code by offering competitive performance. For the Harissa project, a non-native offline byte code compiler that generates C programs has been chosen. Using C as an intermediate language in non-native compilers has some advantages. C makes compiler development safer and quicker, and C compilers are available for all machines.

A Java program contains a set of classes with static reference to each other. The Harissa system loads the entire set of classes at compile time. Its compiler takes the class containing the main method and generates a makefile, a main C file and a C source file for each program class. To generate the C code for different methods, the compiler transforms byte code instructions into C

statements. Thus in summary, Java source code is first compiled into byte code using javac compiler, then Harissa translates the byte code into C code.

In *Harissa: A Hybrid Approach to Java Execution* (1999), the authors provide some benchmark test results of Harissa's aggressive optimizations and its performance relative to JIT compilers. The test result shows that Harissa generated code was faster than JDK 1.2b4, and more efficient than JIT compilers on certain platforms.

Although Java is known to be a portable language, its efficiency is a major concern for application programmers. This article presents a compiler that improves Java's efficiency by keeping the portability factor in tact. Although there is no direct relation between this article and the thesis, it enlightens a new idea of a future research on Java - the efficiency analysis of Java source code in real time business applications.

## **2.7 RESEARCH PROJECTS**

The importance of platform independence directed various research organizations to carry out different research projects on software portability. Some of those projects are directly related to this thesis. This section briefly describes two of the most important ones.

- ***"100% Pure Java" certification program from Sun Microsystems***

The "100% Pure Java" certification program is part of Sun Microsystems' initiative to promote the development of portable applications, applets, beans, and

class libraries written using the Java programming language. There is a subtle distinction between portability and purity. In *100% Pure Java Cookbook for Developers – Rules and Hints for Maximizing Portability*, Sun Microsystems defines "purity" in the following way:

Because portability is so dependent on the functionality of a particular program, we define the concept of "purity." Purity is the aspect of portability that we can measure by looking at the mechanics of how the program uses the platform interface, rather than looking at the program's functionality...Purity is intended to be a measure of only one of the many virtues required of a program. It is not even a perfect measure of that one virtue, portability. It is nonetheless a useful measure; we have found that the purity measures do detect some common portability problems. The purity process does result in better portability; that is our goal, to increase the portability of programs. (1999)

The "100% Pure Java" certification process consists of code analysis and testing by an independent test facility to identify compiled code that meets the "100% Pure Java" requirements. All the certification processes and procedures are explained in a certification guide (<http://java.sun.com/100percent/pjcg.pdf>). When a program or application carries the "100% Pure Java" logo, potential users know it's been thoroughly tested for cross-platform compatibility and portability. This program ensures compliance with the Java specifications and reference implementation by providing a detailed test suite that is known as the "Java Conformance Kit" (JCK). The testing portion of the program is necessarily large and complex because of the nature of the implementations that need to be tested. JCK supports the test part of this project; but unfortunately it is not freely available. According to the project manager Carla Schroer, "We have found that people are not successful with this program without support, and we do not make the JCK available for free." There are approximately 11,000 tests on the current



test suite, and Sun is still not sure that the suite covers the entire language spectrum for portability testing. Sun Microsystems' reluctance in making JCK freely available to independent cleanroom implementers introduced a new project called "Mauve" (<http://sourceware.cygnus.com/mauve>) from Cygnus Corporation. "Mauve" is a collaborative project whose goal is to create a free suite of functional, black box tests for the core Java libraries.

- ***IBM's Jikes project***

"Jikes" (<http://oss.software.ibm.com/developerworks/opensource/jikes/>) is an IBM project. It is a compiler that translates Java source files as defined in the Java language specification (<http://java.sun.com/docs/books/jls/>) into the byte coded instruction set and binary format defined in the Java virtual machine specification (<http://java.sun.com/docs/books/vmspec/>). The Jikes compiler strictly adheres to the Java language specification. It is extremely fast, and it has a built-in dependency analysis tool that allows incremental compilation. It also generates makefile automatically. Anyone can freely acquire the source code of Jikes project and redistribute it. Jikes has already been ported to several platforms. The compiler can find out the basic portability issues of a Java application if the Java language specification is not followed.

## CHAPTER 3

### 3.1 INTRODUCTION OF RESEARCH METHODOLOGY

This section describes the methodology that has been used to perform this research. The research methodology was primarily based on Java's architecture. The test cases were formulated based on generic software portability issues, as well as issues that are specific to the Java language and its class libraries. The goal of the research methodology was to explore how Java faces different portability issues, and to identify where it fails to provide platform independence. Sun Microsystems claims that Java can overcome any of the issues described in chapter 1 that can hinder software portability. The following sections provide reasons for this claim from Cay Horstmann and Gary Cornell of Sun Microsystems.

- *Architecture Neutrality*

The binary code format that the Java system adopts is independent of hardware architectures, operating system interfaces, and window systems. The format of this system-independent binary code is architecture neutral. If the Java run-time platform is made available for a given hardware and software environment, an application written in Java can then execute in that environment without the need to perform any special porting work for that application. The Java compiler does not generate "machine code" for a specific hardware/operating system platform, rather, it generates high level, machine independent byte codes

for a hypothetical machine. The Java virtual machine is a strictly defined machine for which an interpreter is available for each hardware architecture and operating system. The interpreter interprets the byte codes for the virtual machine.

- ***Strict Language Definition***

Architecture neutrality is just one part of software portability. Java technology takes portability a stage further by being strict in its definition of the basic language. Java specifies the sizes of its basic data types and the behavior of its arithmetic operators. Therefore, Java programs are the same on every platform; there are no data type incompatibilities across hardware and software architectures. C and C++ both suffer from the defect of designating many fundamental data types as "implementation dependent". Programmers labor to ensure that programs are portable across architectures by programming to a lowest common denominator.

Java eliminates this issue by defining a standard behavior for its data types across all platforms. Java specifies the sizes of all its primitive data types and the arithmetic behaviors on them. Following are the defined data types of the Java language:

Table 1. Java data types

Byte	8-bit two's complement
Short	16-bit two's complement
Integer	32-bit two's complement
Long	64-bit two's complement
Float	32-bit IEEE 754 floating point
Double	64-bit IEEE 754 floating point
Char	16-bit Unicode character

Thus an *integer* in Java is always a 32 bit *integer*. In C/C++, *integer* can mean a 16 bit *integer*, a 32 bit *integer* or any other size that the compiler vendor likes. Furthermore, in Java, binary data is stored in a fixed format, eliminating the big-endian/little-endian confusion. Strings are saved in a standard Unicode format. Unicode is a code set like ASCII, but because it allows 65,536 different characters rather than the 128 of 7 bit ASCII, Unicode supports essentially all characters of all languages of the world. Seven bit ASCII is only a subset of Unicode.

- ***Java Virtual Machine***

The architecture-neutral and portable language platform of Java technology is known as the Java virtual machine. It is the specification of an abstract machine for which Java programming language compilers can generate code. The Java virtual machine is primarily based on the portable operating system interface (POSIX) specification. POSIX is an industry-standard definition

of a portable system interface. The Java compiler is also written in Java. The Java run-time system is written in ANSI C with a clean portability boundary that is essentially POSIX-compliant. And finally, there are no "implementation-dependent" notes in the Java language specification.

### **3.2 DESCRIPTION OF RESEARCH METHODOLOGY**

Since Sun Microsystems is the only one that claims Java's WORA capability, this research focussed on the Java language, virtual machine, and runtime environment provided by Sun. At the early stage of the research, a wide variety of Java programs (84) were randomly picked from different sources. Those programs were then compiled and executed on different hardware/operating system configurations. The purpose of that approach was to analyze how those programs behave on different system configurations. The test programs generated many compile and runtime errors. Preliminary test data thus helped in finding various Java-specific issues that broke platform independence. Furthermore, it helped in narrowing down different versions of the Java development kit (JDK) and Java runtime environment (JRE) to a specific version for the final test. The next section analyzes the preliminary test data.

### 3.3 ANALYSIS OF PRELIMINARY TEST DATA

- Sun Microsystems has different editions of Java platforms, such as: standard edition, micro edition and enterprise edition. Apart from different editions, Sun also has three different major versions of JDK/JRE sets: 1.0, 1.1 and 1.2. These versions have many sub-versions: for example, the JDK/JRE 1.1 release has sub releases of 1.1.1 through 1.1.8. These different versions of JDK/JRE are neither downward nor upward compatible. In many cases, a program written in one JDK/JRE combination does not even run on a machine that has a different version of JDK/JRE installed on it.

- Programs written in a newer version of JDK are not downward compatible with older versions of JRE because of the introduction of newer methods in newer JDK/JRE sets. For example, JDK/JRE version 1.1.8 is not downward compatible with JDK/JRE version 1.0.2. Additional information can be found at the following web site:

<http://java.sun.com/products/jdk/1.1/docs/relnotes/classlist.html>

- Programs written in an older version of JDK are not upward compatible with newer versions of JRE since different API and/or methods from different API sets have been removed on newer versions of the JDK/JRE set. For example, JDK/JRE version 1.1.8 is not upward compatible with version 1.2.

Additional information can be found at the following web site:

<http://java.sun.com/products/jdk/1.2/compatibility.html#language>

- The event model has also changed from JDK/JRE version 1.0.2 to 1.1.8. It also lacks backward/upward compatibility and breaks Java's portability feature.
- Furthermore, many methods have been deprecated in newer versions of the JDK/JRE set. Although the newer JRE supports the deprecated methods, warning messages appear at compile time when using those deprecated methods and thus they pose concerns in portability.
- Other than the Win32 and Solaris platforms, most of the JDK/JRE portal work on other operating systems is done by third party vendors; for example a software organization called Blackdown Organization ([www.blackdown.org](http://www.blackdown.org)) did the portal of JDK/JRE version 1.1 for Linux; Apple Computers did their portal work for Macintosh systems. Unfortunately, these third party vendors failed to keep up in their portal work with Sun's newer versions of the JDK/JRE set. For example, the last portal work that has been done on both Linux and Macintosh system were based on JDK/JRE version 1.1 as of December 1999.
- Although a specific Java language specification was followed, the portal works vary from platform to platform due to the variations in the internal architecture of hardware and operating systems.

Based on the preliminary data, the final research was narrowed down to a specific version of the JDK/JRE set. Since on most platforms the last portal work was done on JDK/JRE version 1.1.8, that was the version selected for use on this

project. The goal was to prove how well a specific version of the JDK/JRE withstands the portability issues on various platforms.

### 3.4 FINAL TEST METHODS

The following two methods were used to gather data and analyze results.

- Before the emergence of Java, portability referred to a successful compilation and execution of a program on different platforms. The first method was derived to verify Java's platform independence from this traditional software portability standpoint. Although Java's WORA claim indicates that recompilation is not necessary for a Java program to run on any platform, the first method compiled and executed different Java programs on each of the test platforms. The purpose of this test was to analyze the functionality and look and feel of the same source code on multiple platforms. Figure 3 illustrates the first method.

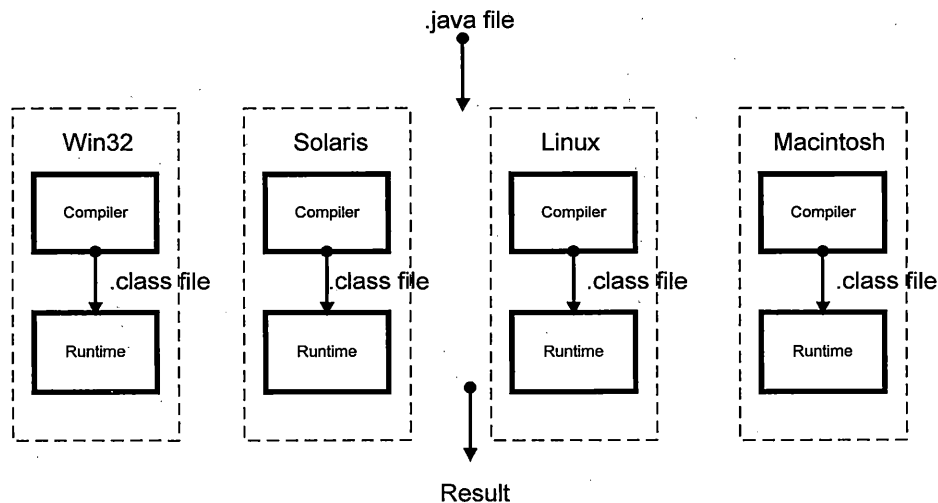


Fig. 3. First method: compile .java files and run .class files on all platforms.



- The second method was used to verify Sun Microsystems' claim of Java's WORA capability. In this method, Java test programs were compiled on a single platform and the resultant class files were then executed on different platforms. The purpose of this test was to verify whether the class files generated on one platform were executable on other platforms. This method also compared the behavior of the same compiled files on different platforms. Figure 4. Illustrates the second method.

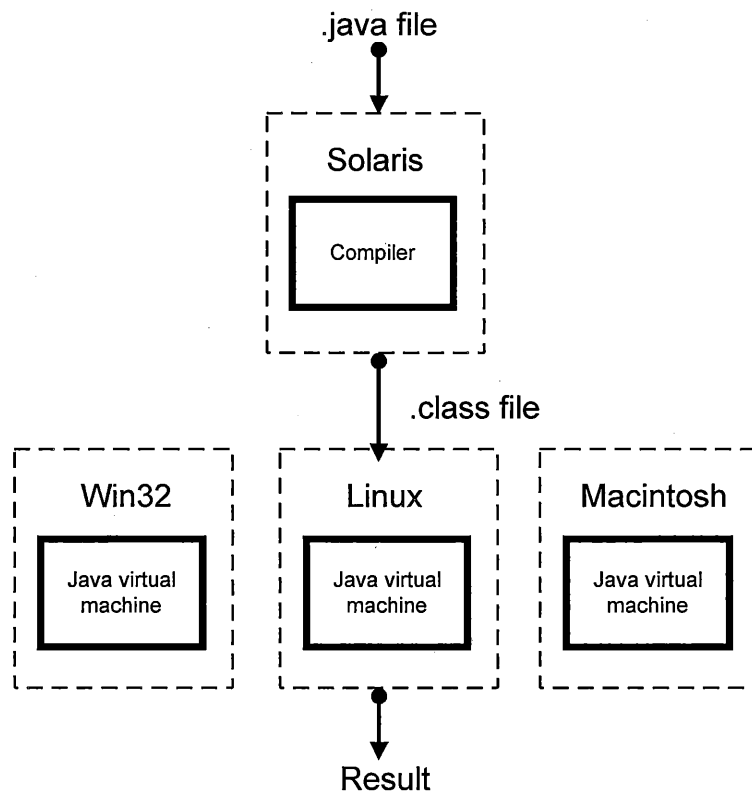


Fig. 4. Second method: compile .java file on a platform, and run the resultant .class files on different platforms.

### 3.5 JAVA TEST CODE

The first task of the final stage of the research was to find enough source code written using JDK version 1.1.8. To achieve this goal, the following steps were performed:

- thorough web search
- thorough research in libraries and bookstores
- contacting several individuals who are actively involved with the Java language. Among them are developers from *JavaSoft* group of Sun Microsystems; writers who wrote articles on Java, and others involved in different Java porting projects.

Once enough source code was gathered, a tool called *JavaPureCheck* was used to verify that the code was written using JDK version 1.1.8. This tool originated from Sun Microsystems, and is primarily used to inspect the portability problems of a Java program on different implementations of the Java platform.

At the final stage, source codes for 73 programs were collected and modified to test each of the portability issues. Later another 21 programs were written without following any portability guidelines to further explore some Java specific portability issues. Forty-two of the 94 programs failed at compile or runtime, or showed functionally wrong output on different platforms.

### **3.6 TYPES OF SYSTEMS, OPERATING SYSTEMS AND VERSIONS OF JDK / JRE USED**

The following platforms and operating systems are used to perform the preliminary and final test:

#### ***Systems***

- Hewlett Packard system, 400 MHz Intel Pentium processor, 64 Meg RAM
- Dell system, 233 MHz AMD K-6 processor, 32 Meg RAM
- IBM system, 133 MHz Intel Pentium processor, 32 Meg RAM
- Sun E450, Dual Processor, 128 Meg RAM
- Apple G3, Macintosh system, 96 Meg RAM

#### ***Operating Systems***

- Win95c
- Win98
- WinNT, Version 4.0 (Service pack 3)
- Solaris, Sparc Version 2.5.7
- Linux, Version 6.1
- Macintosh, Version 8.6

### ***JDK / JRE versions***

JDK/JRE version 1.1.8 was used on all platforms. MRJ 2.4, which is the equivalent of JRE 1.1.8, was used on the Macintosh system. Only the Macintosh system used an integrated development environment (IDE) called Code Warrior, version 3.1, to compile and execute the programs. This version of Code Warrior supports JDK version 1.1.8. The console windows were used to compile and execute programs on all other platforms.

### **3.7 TEST RESULT OVERVIEW**

The following section summarizes the test results. The complete overview can be found in Appendix A in tabular format. The first research method required that each test program be compiled on each of the platforms before being executed. Table 2 provides a list of the types of results that were obtained from performing these tests.

Table 2. Result abbreviation

Successful Compilation	<b>SC</b>
Successful Execution	
Functionally Correct Output	
Unsuccessful Compilation	<b>USC</b>
Compiler Warning	<b>CW</b>
Unsuccessful Execution	<b>USE</b>
Functionally Wrong Output	<b>FWO</b>
Comments in section 3.8	<b>(c)</b>

The summary of the first test results is depicted in the figure below:

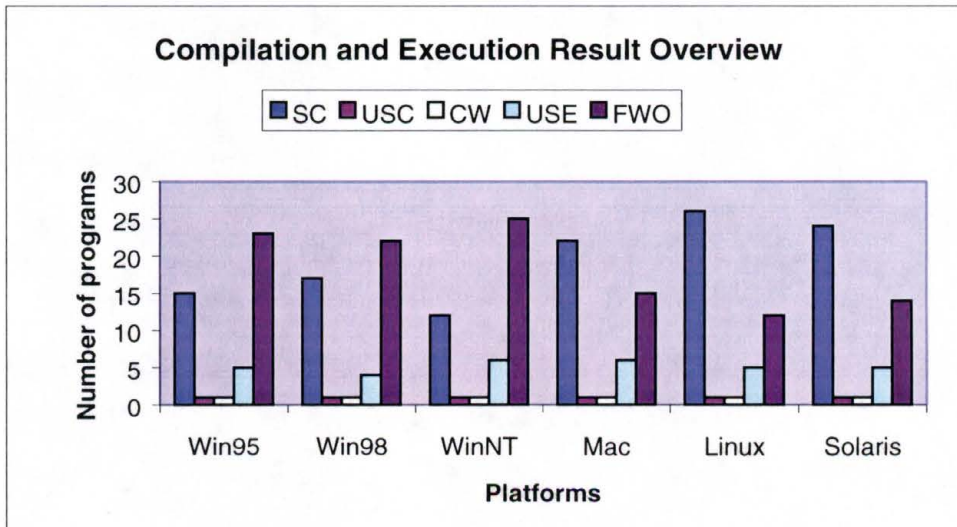


Fig. 5. Compilation and execution results overview.

According to figure 5, the best Java portal work was done under the Linux platform as the highest success rate was attained on this platform. Besides, Linux produced the least number of functionally wrong outputs. Solaris followed Linux

in producing portable code, followed by Macintosh, Win95, Win98 and WinNT. Among the test platforms, WinNT showed the worst portability performance as the highest number of unsuccessful executions occurred on this platform. Besides, the WinNT platform generated the highest number of functionally wrong output from executing the test programs.

The second test method required the test programs to be compiled on a specific platform, and then that compiled code was executed on different platforms. For example, all the test programs were first compiled on an Intel based Windows 95 system. Then the compiled programs were executed on the rest of the test platforms. The following figures show the execution results of the test programs that were compiled on the Intel platform and then run on each of the test platforms.

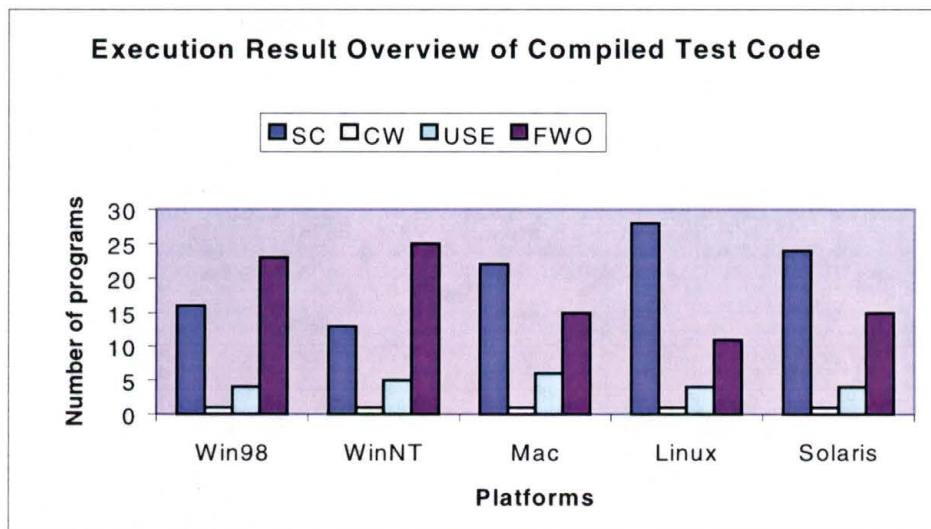


Fig. 6. Execution results of the test programs that were compiled under Win95.

Figure 6 also indicates that the best virtual machine portal work was done for the Linux platform as the highest number of successful executions occurred on this platform. Besides, Linux produced the least number of functionally wrong outputs. Solaris followed Linux in generating portable code, next were the Macintosh and Win32 platforms. This result coincides with the result obtained using the first test method. The rest of the charts also show similar results.

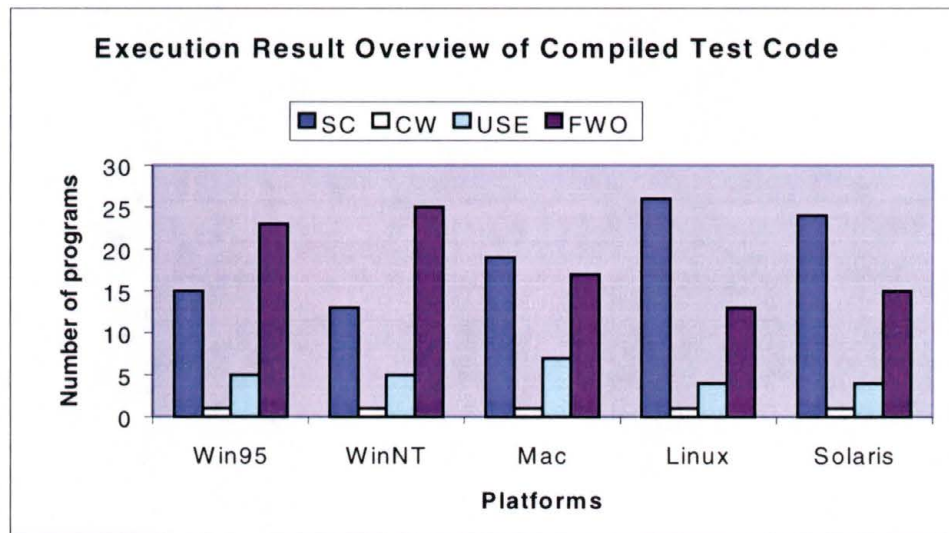


Fig. 7. Execution results of the test programs that were compiled under Win98.

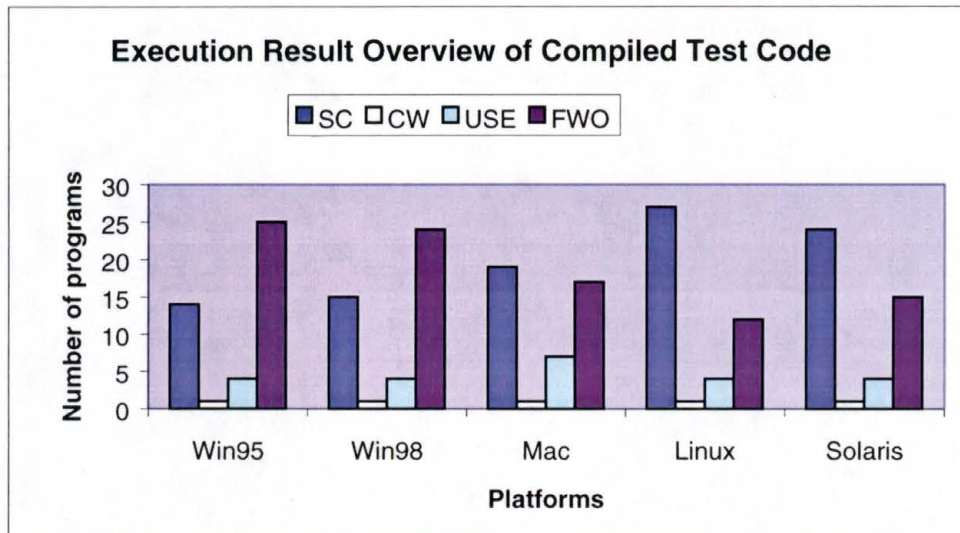


Fig. 8. Execution results of the test programs that were compiled under WinNT.

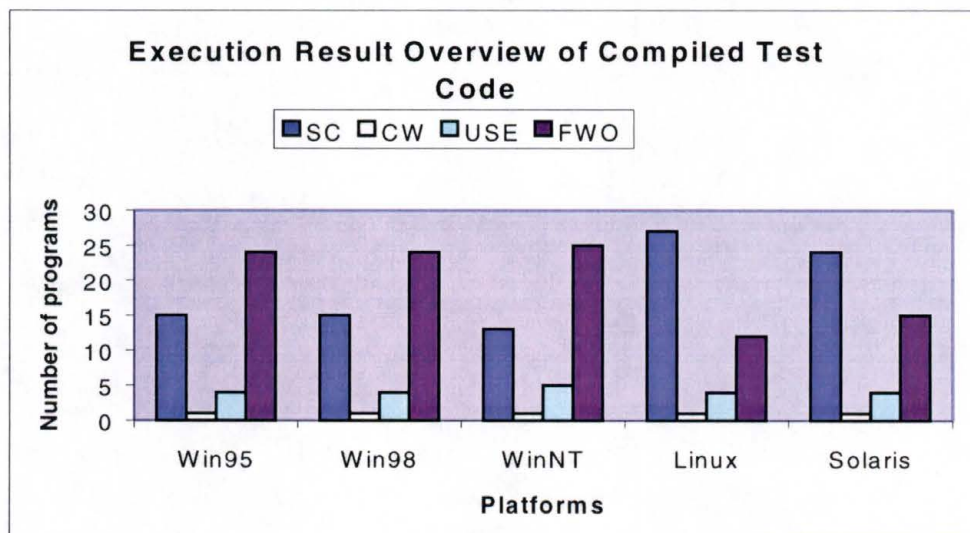


Fig. 9. Execution results of the test programs that were compiled under Mac.



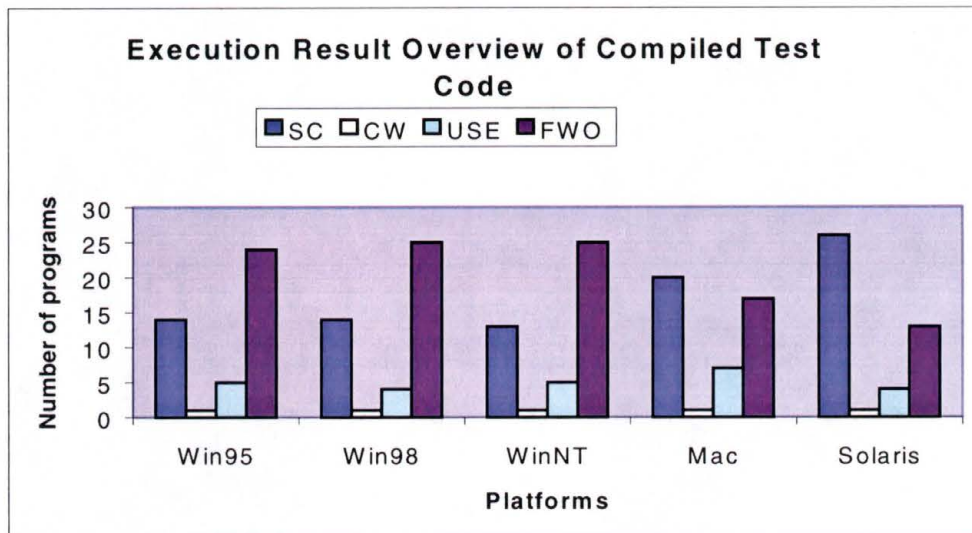


Fig. 10. Execution results of the test programs that were compiled under Linux.

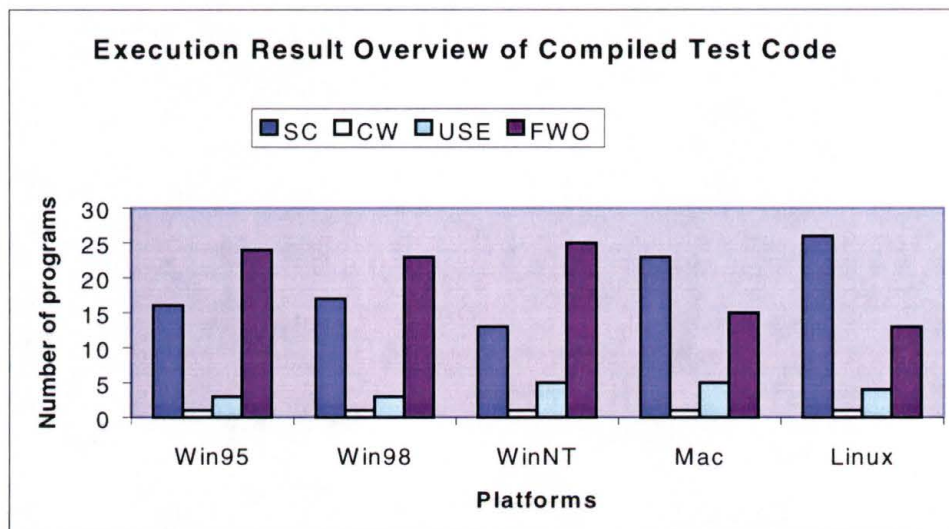


Fig. 11. Execution results of the test programs that were compiled under Solaris.

### 3.8 ANALYSIS OF TEST RESULT

The following section presents the major portability concerns of Java source code by analyzing the test results. The complete list of the test results, and test descriptions can be found in appendix A and B, respectively.

#### *Java Specific issues*

- **Mixing different versions of JDK / JRE**

Different versions (1.0, 1.1 and 1.2) of JDK/JRE are not downward or upward compatible. Hence mixing different versions of JDK/JRE causes compile and runtime errors. For example, the *Calculator* program was used to verify Java's downward compatibility. This test program used a method named `parseDouble(java.lang.String)` that was newly added in JDK 1.2, and not present in JDK version 1.1.8. This program failed to compile under all platforms where JRE 1.1.8 was installed. The following error message was generated at compile time:

```
"Method parseDouble(java.lang.String) not found in class Java.lang.Double
Double.parseDouble(display.getText());

1 error"
```

- **Mixing Event Models**

The event model of JDK 1.1 is different than the event model of 1.0; therefore, mixing these two different event models hinders portability. For

example, in the *EventDemo* program, a method named `handleEvent(java.awt.Event)` was used which was available in event model 1.0, and had been removed from event model 1.1. The purpose of this test was to either prove or disprove downward compatibility of Java's event model. The program generated two warning messages at compile time identifying two deprecated methods, and crashed at run time on all platforms.

Compile time warning message:

```
"EventDemo.java:18: Note: The method boolean action(java.awt.Event,  
java.lang.Object) in class java.awt.Component has been deprecated, and class  
EventDemo (which is not deprecated) overrides it.  
public boolean action(Event event, Object object)EventDemo.java:24: Note: The  
method boolean handleEvent(java.awt.Event) in class java.awt.Component has  
been deprecated.  
return super.handleEvent(event)"
```

Runtime error message:

```
"Exception occurred during event dispatching:  
java.lang.StackOverflowError  
    at java.awt.Component.handleEvent(Compiled Code)  
    at EventDemo.action(Compiled Code)"
```

- **Variance in third party portal work**

The implementations of the Java portal works vary from platform to platform. For example, the *AltTest* program was used to verify how well Java tracks the state of any control key in different platforms. This program prints the

"Up" and "Down" states of the "Alt" key when it is pressed and released respectively inside a window. The result showed that in Win32 platforms, the program lost event notifications at the transition phase of pressing and releasing the "Alt" key. It worked fine in all other test platforms.

Another example was the *ButtonTest* program that showed that MouseEvents generated by clicking a mouse inside a scroll bar were inconsistent across platforms. This program brings up a window with a scrollbar in it. Clicking inside the scrollbar displays the mouse click events in a console window. Under Win32 platforms, the program generated the MOUSE\_PRESSED message for a left button click, and MOUSE\_PRESSED, MOUSE\_RELEASED, and MOUSE\_CLICKED messages for the right button click. In Macintosh, it produced MOUSE\_PRESSED, MOUSE\_RELEASED and MOUSE\_CLICKED messages for a mouse click (note that Macintosh system has a single buttoned mouse). Solaris and Linux platforms on the other hand, did not generate any MouseEvents for a left or a right click.

- **Wrong implementation of certain methods:**

Certain methods do not function according to the Java specification on some platforms. For example, in the *TextFieldTest* program, a method called `KeyEvent.setKeyChar(char)` was used. This method should change lower case characters to upper case. The test showed that the lower case characters did not get converted into upper case on any of the Win32 platforms. The method functioned properly in Linux, Solaris and Macintosh platforms.

- **Using native methods in the program**

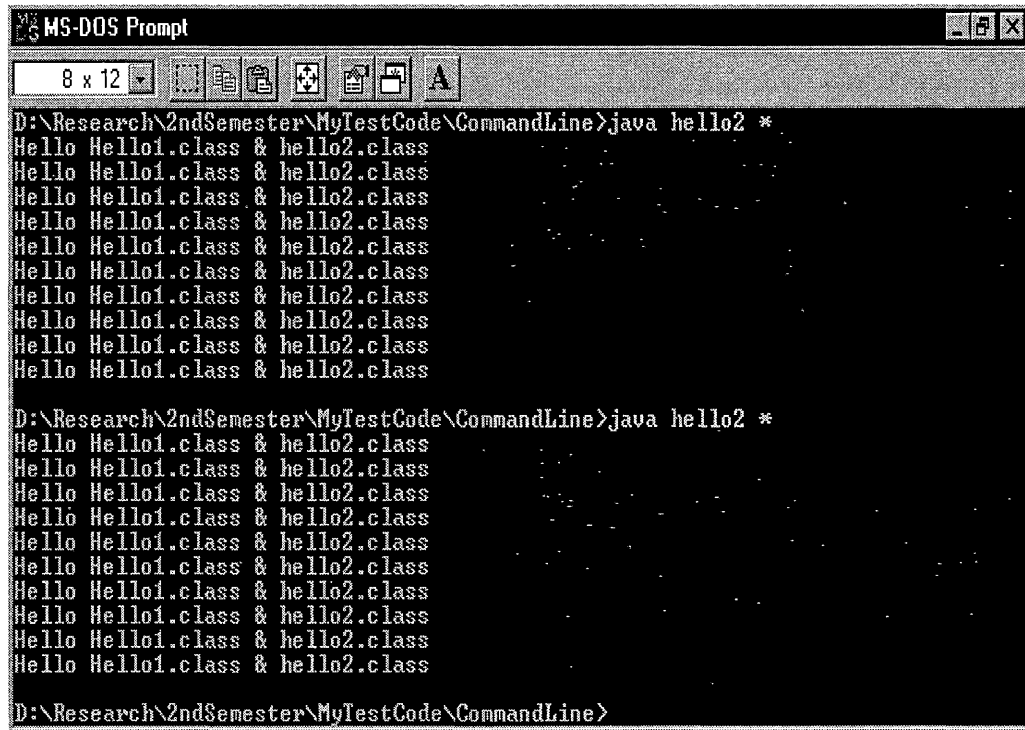
The following two instances break the platform independence of Java:

- dependencies other than the core Java APIs
- dependencies on platform dependent dynamic link libraries

- **Using certain command line programs**

- Command-line programs that use *System.in*, *System.out*, or *System.err* are not portable since not all the Java platforms have the concept of standard input or output streams.
- The Java platforms leave command line processing up to the programmer. However, the syntax and conventions vary from platform to platform, hindering software portability.

For example, the *hello2* program was used to verify the command line processing done by Java on different platforms. When an argument ( \* ) was used while executing the program, it generated different results on different platforms. Some of the captured results are shown below.



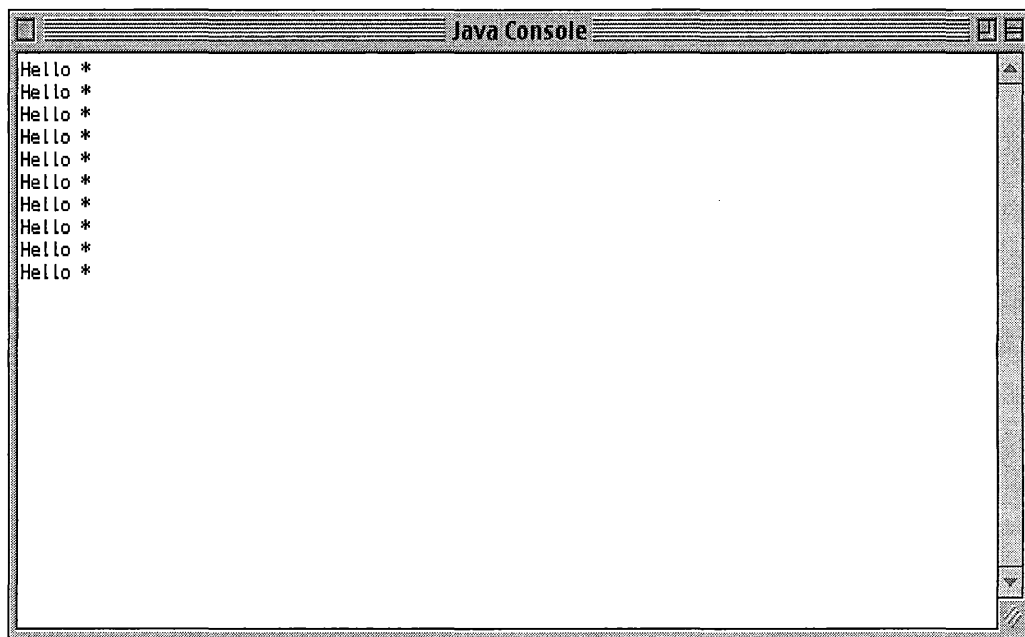
The image shows a screenshot of an MS-DOS Prompt window. The title bar reads "MS-DOS Prompt". The window has a menu bar with "File", "Edit", and "Format" options. Below the menu bar is a toolbar with icons for file operations and a font size dropdown set to "8 x 12". The command prompt shows the following text:

```
D:\Research\2ndSemester\MyTestCode\CommandLine>java hello2 *
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class

D:\Research\2ndSemester\MyTestCode\CommandLine>java hello2 *
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class
Hello Hello1.class & hello2.class

D:\Research\2ndSemester\MyTestCode\CommandLine>
```

Fig. 12. Snapshot of Hello2.java under Win95 platform (correct output).



The image shows a screenshot of a Java Console window. The title bar reads "Java Console". The window has a menu bar with "File", "Edit", and "Format" options. Below the menu bar is a toolbar with icons for file operations and a font size dropdown set to "8 x 12". The console shows the following text:

```
Hello *
Hello *
Hello *
Hello *
Hello *
Hello *
Hello *
Hello *
Hello *
Hello *
```

Fig. 13. Snapshot of Hello2.java under Macintosh (wrong output).

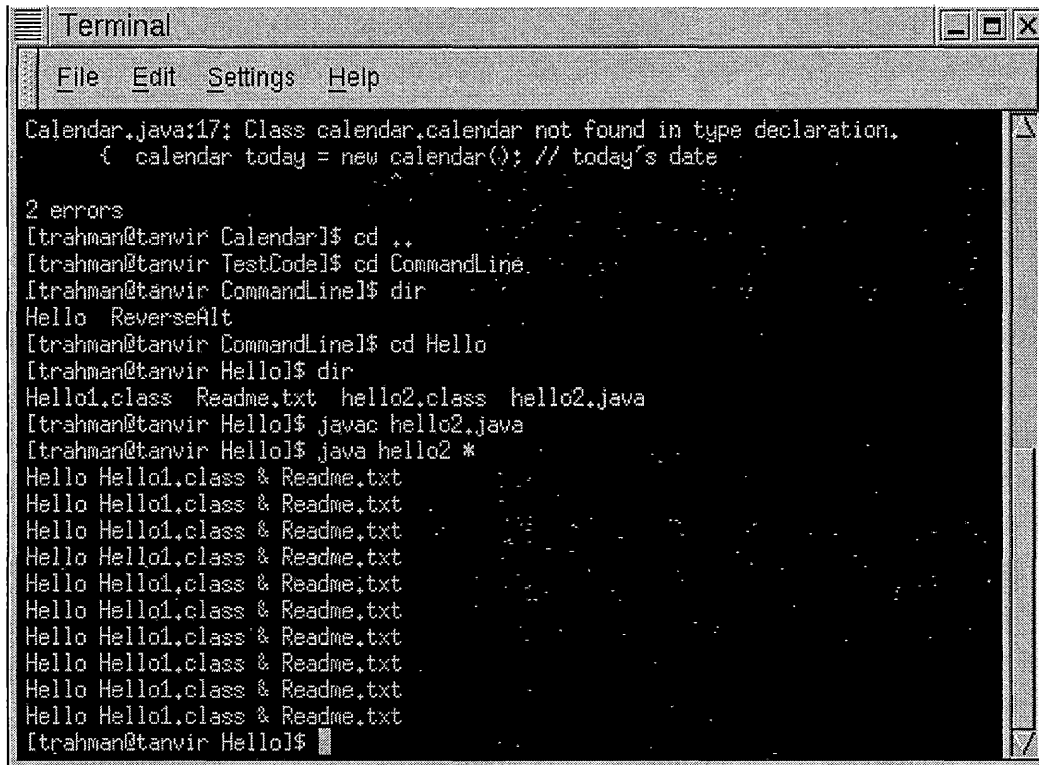
A terminal window titled "Terminal" with a menu bar (File, Edit, Settings, Help). The terminal shows the following sequence of commands and output:  
Calendar.java:17: Class calendar.calendar not found in type declaration.  
 { calendar today = new calendar(); // today's date  
2 errors  
[trahman@tanvir Calendar]\$ cd ..  
[trahman@tanvir TestCode1]\$ cd CommandLine  
[trahman@tanvir CommandLine]\$ dir  
Hello ReverseAlt  
[trahman@tanvir CommandLine]\$ cd Hello  
[trahman@tanvir Hello]\$ dir  
Hello1.class Readme.txt hello2.class hello2.java  
[trahman@tanvir Hello]\$ javac hello2.java  
[trahman@tanvir Hello]\$ java hello2 \*  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
Hello Hello1.class & Readme.txt  
[trahman@tanvir Hello]\$

Fig. 14. Snapshot of Hello2.java under Linux platform (wrong output).

- **Using java.lang.Runtime.exec methods**

The java.lang.Runtime.exec method is not always portable because:

- not all platforms have applications that can be run,
- not all platforms have the notion of *standard input* or *standard output*

For example, in the *ExecTest* program, the "start" command was used inside the exec(String[], String[]) method to open a file named "Hello.txt" in a text editor. The program worked fine on Win95 and Win98 systems. The program hanged at run time without displaying any error message on the WinNT platform although the "start" command is available on the WinNT operating system. Although Linux, Solaris and Macintosh operating systems do not support

the "start" command, the program compiled successfully on all of those platforms. On Linux and Macintosh systems, the program hanged at execution time without any error message. On Solaris, the program properly terminated without displaying any error message.

- **Unicode characters**

Different versions of Java platforms do not render all Unicode characters. For example, the *UnicodeTest* program hard-codes some Unicode numbers, and displays the equivalent Unicode characters on the console window. The purpose of this test was to verify how Java's Unicode character set is handled by different platforms. Different platforms showed different results for Unicode numbers 201C (left quotation mark) and 201D (right quotation mark).

- **Java binary file format**

In Java binary file format, everything is stored as big-endian - most significant byte first. IBM 360, Motorola 68K, Mac PowerPC and most mainframes use big-endian whereas Intel 8080, 8086, 80286, and Pentium use the little-endian convention. This is a major portability issue. The *Readdata* program was used to evaluate this portability concern. This program used a binary file (test.exe ) that was generated by compiling C source code under a Pentium based Win32 system. The *Readdata* program reads the binary executable and writes the data back to an output file (output.dat). As the Pentium processor supports the little-endian scheme, the binary file was in little-endian format. The purpose of



this test was to verify how the Java virtual machine handles little-endian format as it supports big-endian.

Although the same executable file was used in all test systems, the output files generated from different platforms varied. Some platforms did not print "EOF"(End Of File). In general, the output.dat files generated by all the Win32 systems differed from the output.dat files generated by Linux, Solaris and Macintosh systems.

- **Java Epoch Date**

Java's internal clock calculates dates as the time since January 1, 1970. If the system clock is set to an earlier date, Java applications hang or exhibit other unusual behavior on some platforms.

- **Time Zone**

In Java, the list of possible time zones is incomplete, and ambiguously defined. For example, if a programmer wants to express date in the MET timezone (continental European timezone), it gives different results on different platforms. For example, the *MsgLog* program retrieves the current date and time by using MET timezone. The program gave GMT+3h30 : Teheran time on all the test platforms.

- **Using PLAF**

The Pluggable Look And Feel (PLAF) architecture built into the *Swing* classes of the Java 1.2 JDK and the JFC standard extension for JDK 1.1 allows windows, dialogs and other GUI components to take on a distinctive visual identity called a LookAndFeel. But not all PLAFs are available on every platform, and some may only be supported on the operating system the PLAF emulates. For example, the GUI based *PlafTest* application has three buttons to switch among metal, motif and windows look and feel. The purpose of this test was to verify how PLAF is handled on different platforms. Although the *swingall.jar* file was used in the CLASSPATH environment variable, the program window did not switch to Window's look and feel under Linux, Solaris and Macintosh platforms when the "Windows" button was selected.

- **Implementation of Just In Time (JIT) Compiler**

The Win32 Just In Time (JIT) byte code compiler converts virtual machine byte codes to native instructions before execution. This can cause some delay in program startup and class file loading, but it reduces the overall program execution time by a factor of ten. A JIT byte code compiler is included in the Windows version and is used by default. The Macintosh also has its own version of the JIT compiler. But an implementation defect of the JIT compiler often causes errors at runtime. For example, the execution of *Example5* test code generated the following error message at runtime on Win32 and Macintosh

platforms. Since the JIT compiler was not present under Solaris and Linux platforms, the error message did not show up on those platforms.

Error message from JIT compiler:

"nonfatal internal JIT (3.00.078(x)) error 'BinaryNonCommunitive' has occurred.  
in:'Example5.main (Ljava/lang/String;)V: Interpreting method."

### ***General portability issues:***

- **Filename length**

The legal length of a filename varies across platforms. The *FileCopy* program was used to validate this issue. This program copies an existing file into another file. To verify how Java handles long filenames, a file was given a nine character long name. The test program then selected this file and copied it to a new file. All the test platforms copied the file properly.

\*Note: The research failed to validate this issue due to limited numbers of test systems. This could turn out to be a problem in other systems. Lack of varieties of system resources was a major deficiency of this research.

- **Text Case**

Some platforms ignore case when comparing filenames. In Java, if a program has classes or a class and a package whose names differ only by case, the program becomes non-portable to some platforms. The *FileCopy* program was used to prove this concern. The program was successful on all the test platforms, but this issue may turn out to be a real concern on other platforms.

- **Unicode characters as file name**

Using a Unicode character as a file name is not allowable on all platforms.

- **Reserved names**

Some platforms assign special meaning to certain filenames, such as “LPT” or “con.” Use of these filenames as part of a package name for classes that are to be installed as files on those systems becomes a portability issue for Java.

- **File path**

Hard-coded file pathnames are not portable. Hard-coded strings and characters in the source code are also non-portable. The *FileCopy* program was used to copy a text file into another text file. The purpose of this program was to verify how Java handles the hard-coded file path. To verify the issue, a text file named "Hello.txt" was put under a subdirectory, and the path was hard-coded in the Java test code. The program used "\\" as the path separator in the file path. The program succeeded on all Win32 platforms. Since Linux, Solaris and Macintosh use different types of path separators, they failed to copy the file. For example Linux and Solaris use "/" and Macintosh uses ":" as path separators.

- **Path separators and command operators**

Path separator characters are different on different platforms. Unix system uses "/", Windows system uses "\" and Macintosh uses ":". Macintosh uses ":" as an operator for the up command instead of "../" used by Unix. This is also a

portability concern. The *FindDirectory* program moves up a directory and then displays the name of all the subdirectories underneath it. The purpose of this test was to verify how Java handles hard-coded directory move operators. This program used "." to move up one directory. Macintosh failed in this portability test. The program functioned properly on all other platforms.

- **Fonts**

All platforms do not support all the fonts. Hard-coded font names are also not portable. The *FontPicker* program has a list of fonts such as "Times New Roman," "Courier," "Arial," "Dialog," "Kaufmann," etc. The program displays a string using one of those fonts that the user-selects. As "Kaufmann" was not available on Win95, WinNT, Linux, Macintosh, and Solaris, the program did not switch to "Kaufmann" whenever it was selected.

- **Line Termination**

Different platforms have different conventions for line termination in a text file. Different machines have different internal representations of text. Although Java uses Unicode internally which is the solution to this problem, there are programs that need to get text to and from files. Problems arise from reading and writing plain ASCII files since the ASCII standard is not specific about the line termination character. Win32 machines use the "\r\n" sequence, Unix uses "\n", whereas Macintosh uses "\r" as line termination character.

The *LineTerminationTest* program was used to verify how line termination characters are handled on different platforms by the Java virtual machine. This program used a string that was terminated using "\n". The program behaved correctly on Win32, Linux and Solaris systems and failed on Macintosh.

- **Relying on Priorities or Luck**

Relying on priorities or luck in case of multi-threaded programming causes problems since thread scheduling may differ on different platforms. If the program relies on priorities or luck to prevent two threads from accessing the same object at the same time, then it may be non-portable on some platforms. The *ClassLoaderSyncProblem* test code showed that if an instance of a subclass is constructed from its superclass's static initializer, and then classloader loads the superclass and subclass from different threads at the same time, the virtual machine causes deadlock permanently on some platforms. The deadlock occurred on Win95 and WinNT platforms. The code worked correctly on the rest of the test platforms although the successful completion of the two threads varied across platforms.

- **GUI behavior:**

- The physical layout of a program's GUI depends on the sizes of the components that make it up. Any hard-coded positions or sizes of GUI components cause look and feel differences. For example, the *FrameSize* program used a method called `setSize(int,int)` that did not resize its content in a



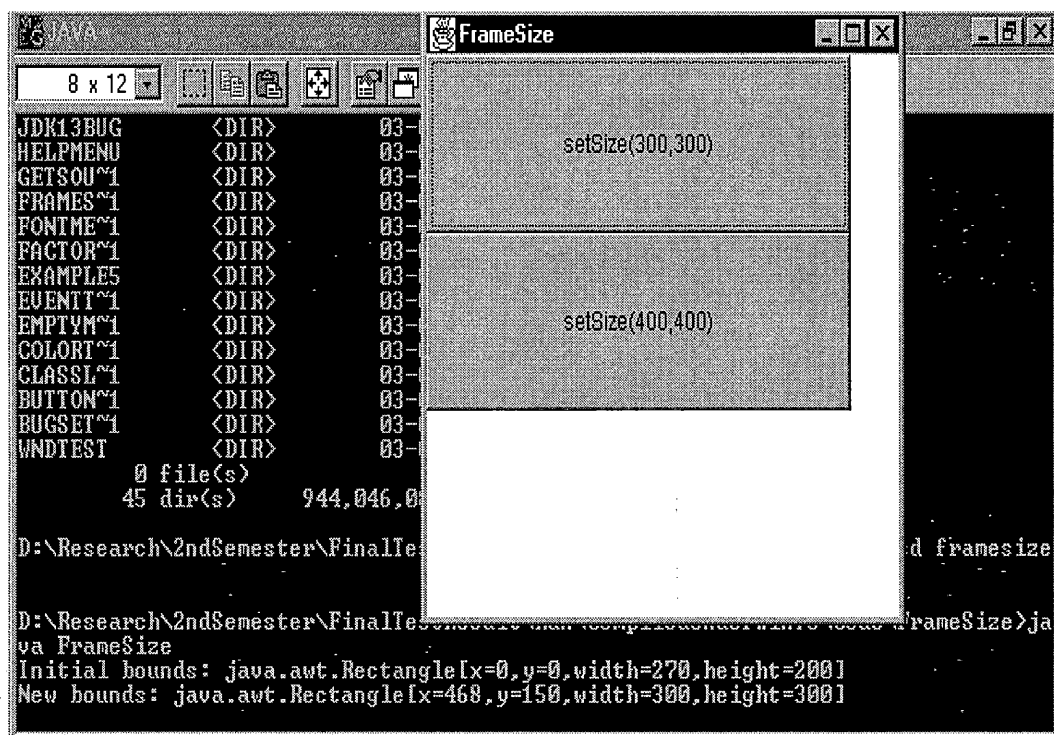


Fig. 16. Snapshot of FrameSize.java program under Win95 platform

(wrong output).

– The size of the screen and the number of available colors may also vary from platform to platform. This makes a display illegible on some platforms. For example, on true-color windows, ImageFilter or PixelGrabber generates gray-scaled color. The *GetSourceBug* test code displays the upper half of a Frame as original image and the lower half as filtered image. The purpose of this test was to verify how the output looks on different platforms. The ImageProducer() method produces grayscale colors in true-color environments. Both red and green are replaced with blue. This phenomenon seems to be restricted to the 32-bit true color setting only. Any lower color depth worked fine. 32-bit is the default on most of the Win98 machines. Other than Win98 systems, the image was



displayed properly on all other platforms. The snapshots of right and wrong images are given below.



Fig. 17. Snapshot of GetSourceBug.java program under Win95 (Left - correct output) and Win98 (Right - wrong output) platforms.

– The size and availability of fonts vary from display to display. In some platforms a specific type and size of font display gets distorted. The *lablebug* test program displayed a string -"Workgroup," using 11point "SansSerif" font. The result showed that the front of character 'W' got clipped on all the Win32

platforms. The string was displayed properly on the other platforms. The snapshots are provided below to show the distinction

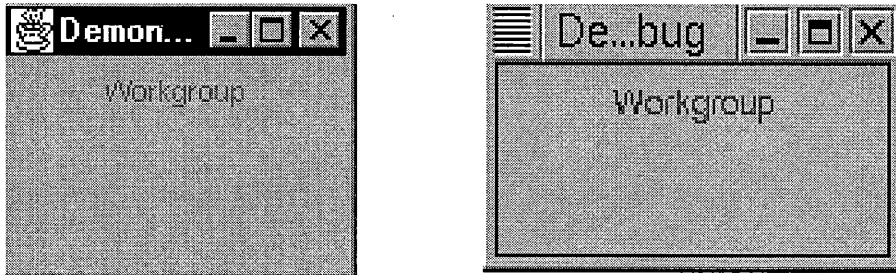


Fig. 18. Snapshot of labelbug.java program under Win95(left - wrong output) and Linux (right - correct output) platform.

- It is non-portable to hard-code a line termination character or text display size.
- There are different adornments around different components on different platforms. For example: on WinNT platform, a TextField object has a single-pixel wide box around it, whereas on Win95, the same component is enclosed in a 3D box.

## **CHAPTER 4**

### **4.1 INFLUENCING FACTORS ON JAVA'S PLATFORM**

#### **INDEPENDENCE**

According to this research, the degree of platform independence of any Java program depends on several factors. Some of these factors are beyond the control of a developer, but most are within his or her control. The Java architecture facilitates the creation of platform-independent software as well as platform specific software. Despite Java's design, the programmer still needs to take some care while writing platform independent code. Many problems are easy to avoid, but some are less obvious. Hence the degree of platform independence depends on how a program is written in Java.

- **Java platform deployment**

The most basic factor of the platform independence of Java programs is the extent to which the Java platform has been deployed on multiple machine architectures. Two things need to happen before running a Java program on a specific computer. First, a port of the Java platform needs to be available for that particular type of hardware and operating system. Secondly, the Java platform vendor must provide an install program to install the platform on a specific computer. Fortunately, along with Sun Microsystems, other third party vendors are also working on Java platform portal work on different machine architectures.

Appendix C contains a list of third party vendors that are involved on Java portal work.

- **Different editions of the Java platform**

The deployment of the Java platform is complicated since not every standard run-time library is guaranteed to be available on every Java platform. The basic set of libraries that are guaranteed to be available on a Java platform is called the *Standard API*. The Java virtual machine accompanied by the *Standard API* (JDK/JRE version 1.2) is called the *Java 2 Platform, Standard Edition*. This edition has the minimum set of Java API libraries that is available for desktop computers (Win32 operating systems) and workstations (Solaris operating systems). Sun also defines API sets for the micro and enterprise editions of the Java 2 platform. The *Micro Edition* (J2ME) of the Java 2 Platform is the subset of the *Standard API* set for consumer and embedded devices. *Java 2 Enterprise Edition* (J2EE) on the other hand, is a superset that includes a set of APIs that are useful in enterprise server environments in addition to the *Standard API* set. Furthermore, there are some standard run-time libraries that Sun considers optional for the standard edition. These optional Sun libraries are called *Standard Extension APIs*. These libraries include services such as telephony, commerce and media such as audio, video and 3D graphics. If a Java program uses libraries from the *Standard Extension API*, it will run anywhere those standard extension libraries are available. But the program will not run on a computer that implements only the basic *Standard API* platform. Given the variety of API editions, the Java 2 Platform hardly represents a single, homogenous execution

environment, and thus breaks Java's WORA capability. The variations in Java platforms also affect JDK/JRE version 1.0 and 1.1.

Java's evolving phase also influences its platform independent nature. Although the Java virtual machine is evolving slowly, the Java API set is changing more frequently. Over time, features are getting added and removed from both the *Standard API* and *Standard Extension APIs*. Furthermore, parts of the *Standard Extension APIs* are also migrating into the *Standard API*. Although the intention was to keep the Java platforms backward compatible, so that they don't break existing Java programs, some changes turned out not to be backward compatible. As features are removed from new versions of the Java platform, programs that depend upon those features do not run any longer. Also some changes are not forward compatible, meaning programs that are compiled for a new version of the Java platform may not necessarily work on an old version because of the added features in the newer version.

- **Native Methods**

The platform independence of a Java program is highly dependent on whether or not a program calls native methods. The most important rule in writing platform-independent Java programs is not to directly or indirectly invoke any native methods that are not part of the core Java API. If it is absolutely necessary to call native methods and at the same time to maintain platform independence, then the required native methods need to be ported to all required platforms and a new API set needs to be defined.

- **Non-standard runtime libraries**

Java platform implementations come from a variety of vendors, and although those vendors must supply the standard run-time libraries, some vendors also supply extra libraries. Those non-standard libraries may call native methods, and thus hinder platform independence.

- **Virtual machine dependencies**

There are two places where variations are allowed in the specification of the Java virtual machines: garbage collection and threading. Since different vendors can implement these two features in different ways, two rules should be followed in order to write platform independent code:

- Do not depend upon timely finalization for program correctness
- Do not depend upon thread prioritization for program correctness

Different vendors use different garbage collection techniques. This flexibility in the Java virtual machine specification means that objects of a particular Java program can be released at completely different times on different virtual machines. Consequently, finalizers that are run by the garbage collector before an object is freed can run at different times on different virtual machines. Therefore, if a finalizer is used to free finite memory resources, such as file handles, the program may run on some virtual machine implementation but may not run on others. On some implementations, the program could run out of the finite resource before the garbage collector gets around to invoking the finalizers that free the resource.

Thread prioritization is another place where variation is allowed in virtual machine implementations. The Java virtual machine specification guarantees that all runnable threads that are at the highest priority will get some CPU time. Furthermore, it also specifies that lower-priority threads will run when higher-priority threads are blocked. However, the specification does not prohibit the lower-priority threads from running when the higher priority threads are not blocked. In some virtual machine implementations, lower-priority threads get some CPU time when the higher-priority threads are not blocked. This often causes problems in program behavior on different platforms. Therefore, to keep the multi-threaded Java programs platform independent, the program should rely on synchronization rather than prioritization.

- **User interface dependencies**

Graphical user interface based programs are most fragile when it comes to portability. The AWT user interface library provides a set of basic user-interface components that map to native components of each platform. The *Swing* library gives advanced components that do not map directly to native components.

Although the AWT and *Swing* libraries make it fairly easy to create a user interface that runs on multiple platforms, they do not necessarily behave correctly on different platforms.

- **Implementation errors**

The implementation errors in the Java language and on the Java portal work are one of the major portability concerns. The vast majority of the problems identified in this thesis are of this nature.

## **4.2 RESEARCH LIMITATIONS**

The success of this research was heavily dependent upon the availability of a wide variety of system environments. Unfortunately, there were not a large number of available system environments. Hence due to lack of systems, various portability issues could not be verified with certainty.

## **4.3 FUTURE RESEARCH**

This research investigated one aspect of Java's WORA claim. Additional research could be performed in the following areas

- Explore Sun Microsystems' JCK test suite to find out how well it identifies portability issues on Java source and binary code.
- Analyze the efficiency of Java source code compared to traditional C or C++ programs.

## **4.4 CONCLUSION**

This research indicates that Java does not live up to its promise in most cases. It is better than traditional C or C++ in terms of portability, but it is not yet all things to all platforms. To generate a portable program using Java, one needs to know what problems to watch out for. Sometimes it is the implementation of the virtual machine and the API that causes difficulty. Sometimes it is the version



incompatibilities of JDK/JRE set that should receive special attention. Sun Microsystems' "100% Pure Java" certification program was initiated with a vision to flawlessly run any Java program on any Java-compatible platform or device. But the developers need to keep in mind that "100% Pure Java" is not an end-all or cure-all for developing cross-platform Java applications. This research thus concludes that

- Java version 1.1.8 does not provide full WORA capability. Although the strict language definition and the virtual machine of Java helps in generating more portable programs than that of C or C++, still a perfect WORA world for Java developers is not here yet. Forty-two of the first 94 test programs disproved Sun Microsystems' claim of Java's WORA capability.

- Without following additional development guidelines, it is not possible to write portable Java programs. Again, the Java architecture, especially, its class file format and the Java platform layer (virtual machine and API) handle several traditional portability concerns, but still there are some platform specific issues for which a development guideline needs to be followed to generate portable programs. Of the first 94 programs that were used on the final stage, 9 were written without following any portability guidelines. These programs are both IO and graphical user interface based. Five of the 9 programs failed in portability testing.

- Java version 1.1.8 does not provide full GUI portability. Especially the AWT libraries of the Java language pose a threat to generating portable programs. Of the 42 final test programs, 27 were solely used to test the portability concerns

of graphical user interfaces. These programs generated functionally wrong outputs on at least one of the test platforms. The rest of the 15 programs were used to test the portability concerns of IO functionality of Java. In a few cases, these programs caused compile and runtime errors, and mostly generated functionally wrong outputs on at least one of the test platforms.

This thesis explored some of the portability issues that affect Java programs. Following some basic guidelines to address those issues, in most cases, will resolve most of the portability concerns. Since different versions of the JDK/JRE set are not downward or upward compatible, a Java program should be written for a particular version of JDK/JRE. The event model of JDK/JRE version 1.0 differs from the event model of 1.1, and they are also incompatible with each other. Hence, event models should not be mixed in the implementation of a Java program. Invoking native methods or depending upon anything other than the core Java API set also hinders cross platform independence. If it is imperative to use a native method then it should be rewritten using the Java programming language and should be ported to the platforms where the program is intended to run. Variance in third party portal work also poses a threat to Java's portability. Furthermore, there are some methods in core API set that are not portable by nature. For example, `java.lang.Runtime.exec(String[],String[])` method. This method is used to execute an application from a Java program. Since not all platforms support all the applications, it often hinders portability. In case of a multi-threaded program relying on priorities or luck should be avoided. In order to generate platform independent code, platform specific constants also

need to be avoided. Hard-coded strings, characters, file path names and font names make a program non-portable. In case of a GUI application, a layout manager should be used instead of hard-coding element or screen size, position, and color. Since the Unicode character set is not supported by all platforms, it is wise to use ASCII as default text on GUI components. It is not possible to generate WORA capable software by merely following these guidelines, but adherence to these guidelines will reduce the difficulty involved in porting the software.

## APPENDIX A

### TEST RESULT OVERVIEW

Table 3. Result abbreviation

Successful Compilation Successful Execution Functionally Correct Output	SC
Unsuccessful Compilation	USC
Compiler Warning	CW
Unsuccessful Execution	USE
Functionally Wrong Output	FWO
Comments in section 3.8	(c)

The following metrics gives an overview of the test result.

**[a] Test Description:** Take a specific type of source code, then compile and run it on different types of system configuration.

Table 4. Result overview of test 1

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	SC
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	USE	USE	USE
FindDirectory	SC	SC	SC	USE	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	FWO	SC	SC	FWO
BugSetLocation	FWO	SC	FWO	SC	FWO	FWO
ButtonTest	SC	SC	SC	SC	FWO	FWO
ClassLoaderSyncProblem	USE	SC	USE	SC	SC	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 4. Result overview of test 1

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	SC	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	SC(c)	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	SC	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 4. Result overview of test 1

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	SC	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC	FWO	SC
TrivialApplication	SC	SC	SC	FWO	FWO	SC
TypeAhead	FWO	FWO	FWO	SC	SC	FWO
WinDiaFocus	SC	SC	SC	FWO	SC	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

**[b] Test Description:** Compile source codes in a specific type of system configuration; then run the byte codes on different systems configuration.



Table 5. Test configuration 1- Compiled in Intel system under Windows '95 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	FWO
Hello	SC	SC	SC	FWO	SC	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	FWO	SC	SC
FindDirectory	SC	SC	SC	USE	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	SC	FWO	SC	FWO
BugSetLocation	FWO	SC	FWO	SC	FWO	FWO
ButtonTest	SC	SC	SC	SC	FWO	FWO
ClassLoaderSyncProblem	USE	SC	SC	SC	SC	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 5. Test configuration 1- Compiled in Intel system under Windows '95 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	SC	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	SC(c)	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	SC	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 5. Test configuration 1- Compiled in Intel system under Windows '95 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	SC	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC(c)	SC	SC
TrivialApplication	SC	SC	SC	SC	SC	SC
TypeAhead	FWO	FWO	FWO	SC(c)	FWO	FWO
WinDiaFocus	SC	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

Table 6. Test configuration 2: Compiled in Intel system under Windows '98 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	FWO
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	FWO	SC	SC
FindDirectory	SC	SC	SC	USE	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	SC	FWO	SC	FWO
BugSetLocation	SC	SC	FWO	SC	FWO	FWO
ButtonTest	SC	SC	SC	FWO	FWO	FWO
ClassLoaderSyncProblem	USE	SC	SC	SC	SC	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 6. Test configuration 2: Compiled in Intel system under Windows '98 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	USE	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	FWO	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	SC	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 6. Test configuration 2: Compiled in Intel system under Windows '98 OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	FWO	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC	SC	SC
TrivialApplication	SC	SC	SC	SC	SC	SC
TypeAhead	FWO	FWO	FWO	SC	FWO	FWO
WinDiaFocus	SC	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

Table 7. Test configuration 3 - Compiled in Intel system under Windows NT OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	SC	FWO
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	FWO	SC	SC
FindDirectory	SC	SC	SC	USE	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	FWO	FWO	SC	FWO
BugSetLocation	FWO	SC	FWO	SC	FWO	FWO
ButtonTest	SC	SC	SC	FWO	FWO	FWO
ClassLoaderSyncProblem	FWO	SC	SC(c)	SC	SC	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 7. Test configuration 3 - Compiled in Intel system under Windows NT OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	USE	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	FWO	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	FWO	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	SC	FWO	FWO	SC	SC	SC
TestKeyListener	FWO	SC	SC	FWO	SC(c)	SC(c)



Table 7. Test configuration 3 - Compiled in Intel system under Windows NT OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	FWO	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC	SC	SC
TrivialApplication	SC	SC	SC	SC	SC	SC
TypeAhead	FWO	FWO	FWO	SC	FWO	FWO
WinDiaFocus	SC	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

Table 8. Test Configuration 4: Compiled in Apple system using Mac OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	FWO
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	FWO	SC	SC
FindDirectory	SC	SC	SC	SC	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	SC	FWO	SC	FWO
BugSetLocation	FWO	SC	FWO	FWO	SC	FWO
ButtonTest	SC	SC(c)	SC	SC(c)	FWO	FWO
ClassLoaderSyncProblem	FWO	FWO	SC	SC	SC(c)	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 8. Test Configuration 4: Compiled in Apple system using Mac OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	SC	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	FWO	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	SC	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 8. Test Configuration 4: Compiled in Apple system using Mac OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	SC	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	FWO	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC(c)	SC	SC
TrivialApplication	SC	SC	SC	FWO	SC	SC
TypeAhead	FWO	FWO	FWO	SC	FWO	FWO
WinDiaFocus	SC	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

Table 9. Test Configuration 5: Compiled in Intel system under Linux OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	FWO
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	USE	SC	USE	USE	USE	USE
LineTerminationTest	SC(c)	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	SC	SC	SC
FindDirectory	SC	SC	SC	USE	SC	SC
FontPicker	SC	SC	SC	SC	SC	SC
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	FWO	FWO	SC	FWO
BugSetLocation	SC	SC	FWO	FWO	FWO	FWO
ButtonTest	SC	SC(c)	SC	FWO	SC(c)	FWO
ClassLoaderSyncProblem	FWO	FWO	FWO	SC	SC	SC
ColorTest	FWO	FWO	FWO	FWO	USE	USE

Table 9. Test Configuration 5: Compiled in Intel system under Linux OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	USE	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	SC(c)	SC	SC
Jitbug	USE	USE	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	FWO	FWO	FWO	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	SC
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	SC	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 9. Test Configuration 5: Compiled in Intel system under Linux OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	SC	FWO	FWO	SC	SC	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	FWO	SC(c)	SC
TrivialApplication	SC	SC	SC	SC	SC	SC
TypeAhead	FWO	FWO	FWO	SC(c)	FWO	FWO
WinDiaFocus	FWO	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

Table 10. Test configuration 7: Compiled in Sun Workstation under Solaris OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
Calculator	USC	USC	USC	USC	USC	USC
EventDemo	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE	CW/USE
Readdata	SC	SC	SC	FWO	FWO	SC
Hello	SC	SC	SC	FWO	FWO	FWO
ExecTest	SC	SC	USE	USE	USE	USE
LineTerminationTest	SC	SC	SC	FWO	SC	SC
FileCopy	SC	SC	SC	FWO	SC	SC
FindDirectory	SC	SC	SC	SC	SC	SC
FontPicker	FWO	SC	FWO	FWO	FWO	FWO
PlafTest	SC	SC	SC	FWO	FWO	FWO
MsgLog	FWO	FWO	FWO	FWO	FWO	FWO
UnicodeTest	FWO	FWO	FWO	FWO	FWO	FWO
AltTest	FWO	FWO	FWO	SC	SC	FWO
BugSetLocation	SC	SC	FWO	SC	FWO	FWO
ButtonTest	SC	SC	SC	SC	FWO	FWO
ClassLoaderSyncProblem	FWO	SC	SC	SC	SC	SC
ColorTest	SC	SC	FWO	FWO	USE	USE



Table 10. Test configuration 7: Compiled in Sun Workstation under Solaris OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
EmptyMenubarBug	FWO	FWO	FWO	FWO	SC	SC
EventTest	FWO	FWO	FWO	SC	FWO	FWO
Example5	USE	USE	USE	USE	SC	SC
Factors42	USE	USE	USE	SC	USE	USE
FrameSize	FWO	FWO	FWO	FWO	SC	SC
GetSourceBug	SC	FWO	SC	SC	SC	SC
HelpMenu	FWO	FWO	FWO	SC(c)	SC	SC
Jitbug	FWO	FWO	USE	USE	SC	SC
Labelbug	FWO	FWO	FWO	SC	SC	SC
ListAction	FWO	FWO	FWO	FWO	SC	SC
ModalDialogTest	FWO	FWO	FWO	FWO	SC	FWO
ModifiersTest	SC	SC	SC	SC	SC	SC
ScrollPaneTest	SC	SC	FWO	SC	FWO	FWO
SetSizeBug	FWO	FWO	FWO	SC	SC	SC
Test2	FWO	FWO	FWO	SC	SC	SC
TestCase	FWO	FWO	FWO	SC	SC	SC
TestKeyListener	SC	SC	SC	FWO	SC(c)	SC(c)

Table 10. Test configuration 7: Compiled in Sun Workstation under Solaris OS

Source Code	Win95	Win98	WinNT	Macintosh	Linux	Solaris
TestPopup	FWO	FWO	FWO	SC	SC	SC
TestScrollBar	FWO	FWO	FWO	SC	SC	SC
TestTextField	FWO	FWO	FWO	SC	FWO	FWO
TextFieldTest	FWO	FWO	FWO	SC	SC	SC
TFBehavior	FWO	FWO	FWO	SC	SC	SC
TrivialApplication	SC	SC	SC	FWO	SC	SC
TypeAhead	FWO	FWO	FWO	SC	FWO	FWO
WinDiaFocus	SC	SC	SC	FWO	FWO	FWO
WindowTest	FWO	FWO	FWO	SC	SC	SC
WndTest	FWO	FWO	FWO	SC	SC	SC

## APPENDIX B

### A COMPLETE TEST ANALYSIS

The following section describes the test codes and analyzes the result for different platforms.

#### AWT / Swing Based Programs

##### 1. EmptyMenubarBug

*Type*            AWT

*Description*    The Menubar without any menus become invisible in some platforms. The purpose of this test was to verify this behavior under all platforms.

The test code brings up a frame with a menu bar that initially has 2 menus in it.

After a few seconds, the program removes the menus from the menubar.

#### *Analyze Result*

WIN95            Menu bar disappeared with the menus.

WIN98            Menu bar disappeared with the menus.

WINNT            Menu bar disappeared with the menus.

LINUX            The menu bar did not disappear.

SOLARIS          The menu bar did not disappear.

MAC              Exhibited a different problem. No menu bar showed on the actual program window, and the menus got added to Apple menu bar.

The menus disappeared from the apple menu bar after the given time.

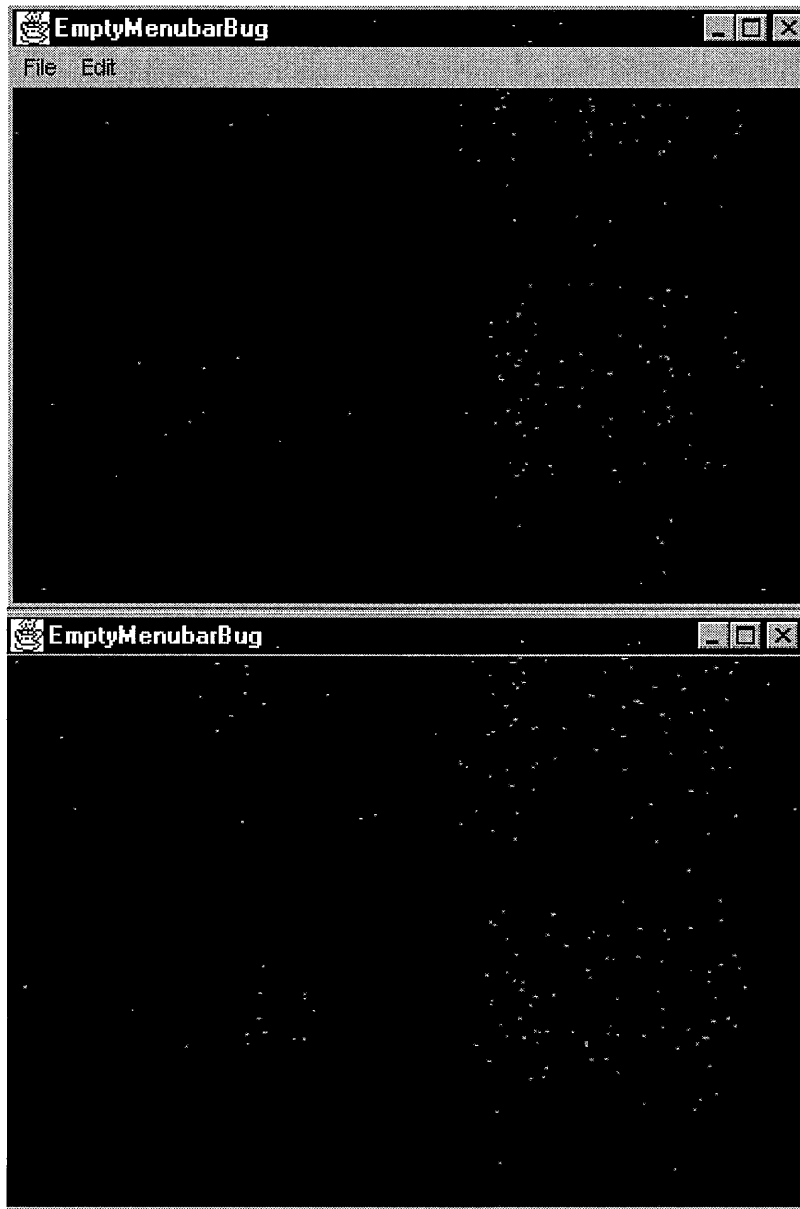


Fig. 19. Snapshot of EmptyMenubarBug.java program under Win95 platform  
(wrong output).

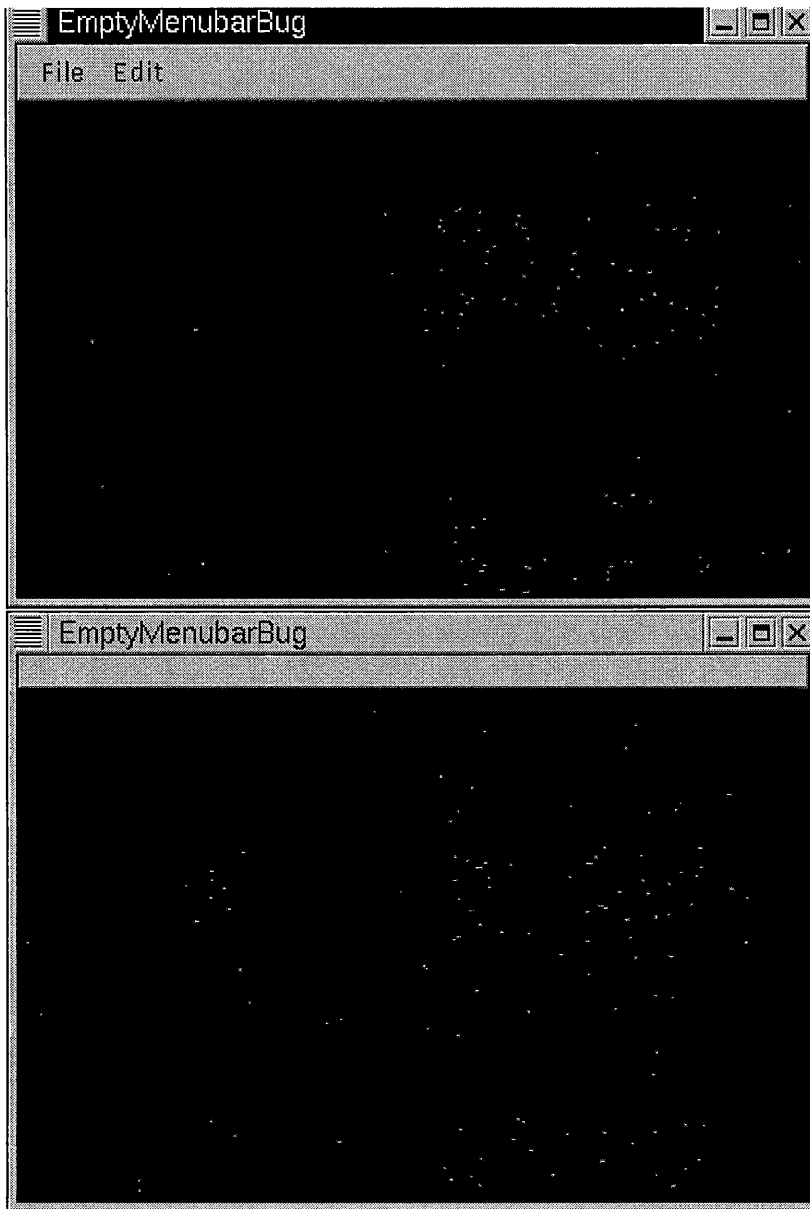


Fig. 20. Snapshot of EmptyMenubarBug.java program under Linux platform  
(correct output).

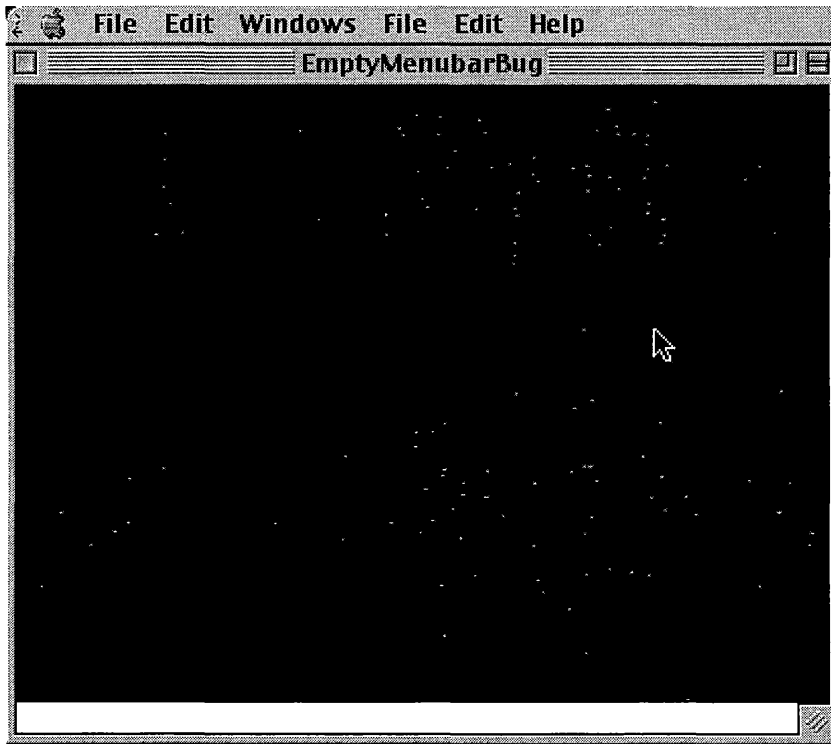


Fig. 21. Snapshot of EmptyMenubarBug.java program under Macintosh platform  
(wrong output).

## 2. EventTest

**Type** AWT

**Description** The `e.getModifiers()` method always returns 0 when ActionEvents are generated on Buttons or on TextFields on some platforms. The purpose of this test program was to verify the behavior across all platforms. This program brings a window with a text field and a button in it. Typing in the text field and pressing "enter" key returns a modifier value. Pressing the button also returns a modifier value.

**Analyze**      The ActionEvents are being created with the wrong constructors that do not pass the modifiers.

Constructor methods:

- ActionEvent(Object, int, String)  
Constructs an ActionEvent object with the specified source object.
- ActionEvent(Object, int, String, int)  
Constructs an ActionEvent object with the specified source object.

Workaround of this problem is to use MouseListener.

**Result**

WIN95	Returned 0 as a modifier value.
WIN98	Returned 0 as a modifier value.
WINNT	Returned 0 as a modifier value.
LINUX	Returned 0 as a modifier value.
SOLARIS	Returned 0 as a modifier value.
MAC	Returned right modifier value.

**3. Bug\_setLocation**

**Type:**      AWT

**Description**    The getLocationOnScreen( ) method often reports stale information when it is invoked immediately after setLocation(int, int ) call on a window object. The purpose of this test was to verify this behavior on different

platforms. The test program brings up a window with a button in it. When the button is pressed the window moves to a different location on the screen.

**Analyze** The problem was reproduced on multiple platforms. It seems like the setLocation(int, int) method should not return until the window has actually moved . A PostMessage to SendMessage conversion in setLocation(int,int ) method may fix the problem.

**Result**

WIN95	Behaved as described. But the occurrence is very infrequent.
WIN98	Worked fine.
WINNT	Behaved as described.
LINUX	Behaved as described.
SOLARIS	Behaved as described.
MAC	Worked fine.

**4. TrivialApplication**

**Type** IO / AWT

**Description** In some platform, the file system doesn't handle illegal characters in text encoding. This test application uses a FileDialog to select a file. A test file is created using illegal characters. When that file is selected from the Java test code, it tries to opens the file with a FileReader which fails on a platoform with a FileNotFoundException.

**Analyze** If the file system encounters a file with illegal characters (eg, \ / etc) it converts the illegal chars to 0xF0xx. Somewhere in the Java encoding,



these special characters get converted to '?' (0x3F). This makes it impossible to open the file with an InputStream or Reader object.

### ***Result***

WIN95	Worked fine.
WIN98	Worked fine.
WINNT	Worked fine.
LINUX	Worked fine.
SOLARIS	Worked fine.
MAC	Behaved as described.

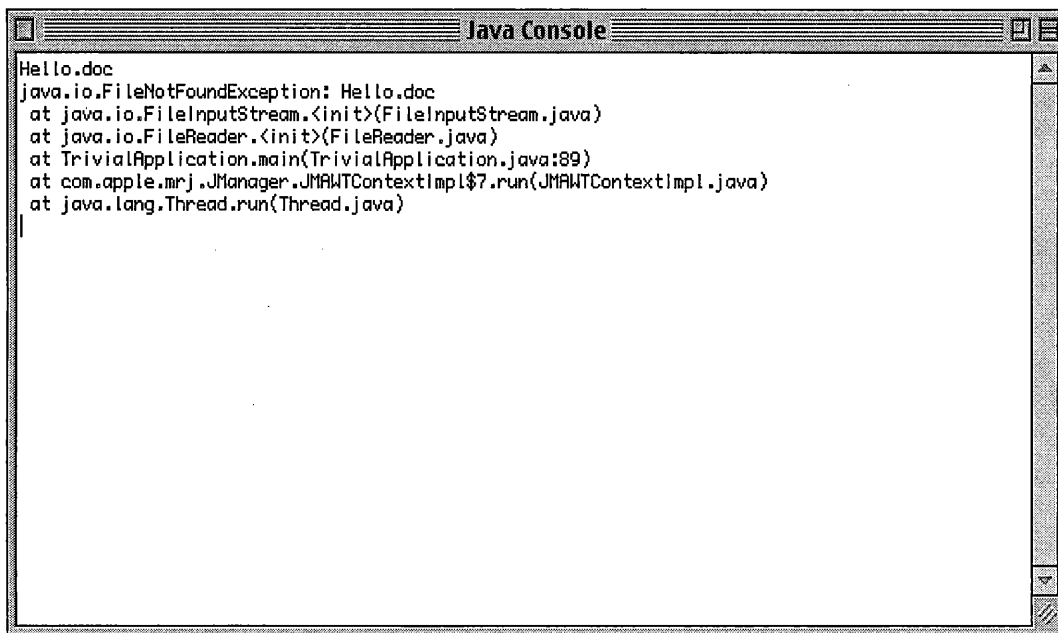


Fig. 22. Snapshot of TrivialApplication.java program under Macintosh platform  
(wrong output).

## 5. HelpMenu

**Type** AWT

**Description** The SetHelpMenu() method does not work consistently on different platforms. The test program brings up a window which has a button called "TestCode" in it. The window has a menubar with two menus: "File" and "Help." Pressing the "TestButton" adds a new menu called "Java" on the menubar, and then sets the "Help" menu. In some platforms an extra blank space gets added in between "File" and "Help" menu, every time a new menu gets added and SetHelpMenu(Menu) is called.

**Analyze** Workaround may be to remove the "Help" menu prior to adding the new menu; and then call setHelpMenu(Menu) method.

```
remove(helpMenu);  
add(newMenu);  
setHelpMenu(helpMenu);
```

### **Result**

WIN95 "Help" menu appeared close to "File" menu. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added menu gets placed in between "File" and "Help" menus. A blank space got added in between "File" and newly added menu.

WIN98 "Help" menu appeared close to "File" menu. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added menu gets placed in between "File" and "Help"

menus. A blank space got added in between "File" and newly added menu.

WINNT

"Help" menu appeared close to "File" menu. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added menu gets placed in between "File" and "Help" menus. A blank space got added in between "File" and newly added menu.

LINUX

"Help" menu showed up at the farthest right of the menu bar. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added "Java" menu got placed in between "File" and "Help" menu leaving no empty space between "File" and newly added menu.

SOLARIS

"Help" menu showed up close to the "File" menu. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added "Java" menu got placed in between "File" and "Help" menu leaving no empty space between "File" and newly added menu.

MAC

Exhibited a different problem. The menus got appended in Apple menu bar. No menu bar showed on the actual program window. Pressing the "Test" button added "Java" menu by shifting "Help" menu to the right. The newly added "Java" menu got placed in between "File" and "Help" menu leaving no empty space between "File" and newly added menu.

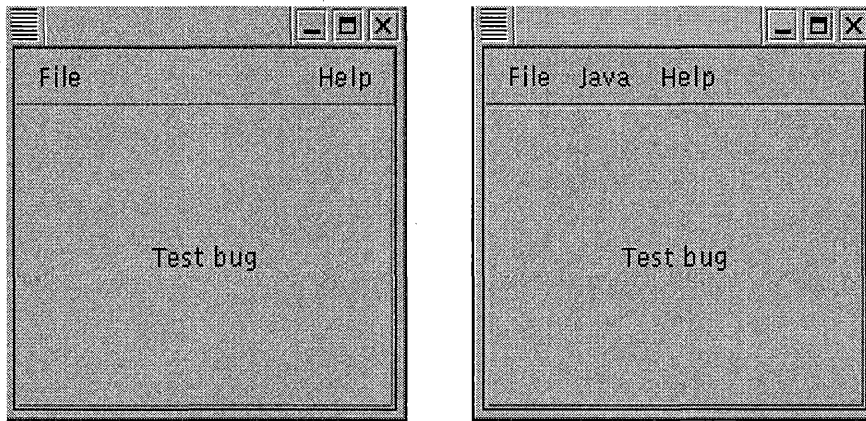


Fig. 23. Snapshot of HelpMenu.java program under Linux platform  
(correct output).

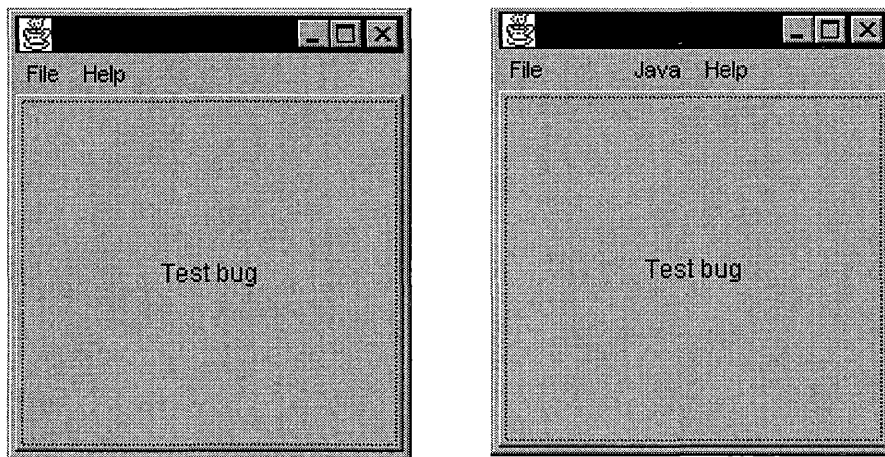


Fig. 24. Snapshot of HelpMenu.java program under Win95 platform  
(wrong output).

## 6. ListAction

**Type** AWT

**Description** In some platforms, pressing "Enter" or "Return" key after selecting an item from a list doesn't generate "actionPerformed" notification as specified in the Java language specification. This program brings up a window with a list that has multiple items in it. Selecting an item and then pressing "Return" key does not generate similar event notification on all platforms.

**Analyze** The JDK documentation for java.awt.List states that the "AWT also generates an action event when the user presses the return key while an item in the list is selected." The test program showed that the actual implementation varied from the specification.

### **Result**

WIN95	[a] Single click generated "itemStateChanged" notification event. [b] Double click generated "itemStateChanged" followed by "actionPerformed" notification event. [c] Pressing "Return" key after selecting an item did not generate any event message.
WIN98	[a] Single click generated "itemStateChanged" notification event. [b] Double click generated "itemStateChanged" followed by "actionPerformed" notification event. [c] Pressing "Return" key after selecting an item did not generate any event message.
WINNT	[a] Single click generated "itemStateChanged" notification event.

[b] Double click generated "itemStateChanged" followed by "actionPerformed" notification events.

[c] Pressing "Return" key after selecting an item did not generate any event message.

#### LINUX

[a] Single click generated "itemStateChanged" notification event.

[b] Double click generated "itemStateChanged" followed by "actionPerformed" notification events.

[c] Pressing "Return" after selecting an item generated "actionPerformed" message.

#### SOLARIS

[a] Single click generated "itemStateChanged" notification event.

[b] Double click generated "itemStateChanged" followed by "actionPerformed" notification events.

[c] Pressing "Return" after selecting an item generated "actionPerformed" message.

#### MAC

[a] Single click generates "itemStateChanged" message twice.

[b] Double click generates "itemStateChanged" message twice followed by a single "actionPerformed" message.

[c] Pressing "Return" key after selecting an item doesn't generate any message.

## 7. **ModalDialogTest**

*Type*            AWT

*Description*    In some platforms, modal dialog box blocks input not only to its parent window, but also to all frame windows. The test case creates two frame windows. Both of the frame windows have a single button in them. Clicking the button of one of the frame windows causes a message to appear on the console window. Clicking the button of the other frame window creates a modal dialog box. As it appears, all frame windows get blocked when the modal dialog box gets created. Clicking on the button of the other frame window does not generate any message.

### *Analyze Result*

WIN95	Caused a global block.
WIN98	Caused a global block.
WINNT	Caused a global block.
LINUX	Did not cause a global block. The other frame could still receive inputs even after the modal dialog was up.
SOLARIS	Caused a global block.
MAC	Caused a global block.

## 8. **ModifiersTest**

*Type*            Swing

*Description*    Modifiers occasionally behaves improperly with events on some platforms. This test code brings up a window with a text field in it. The code

requires the user to click in the text field, hold down the Shift key, and quickly hit the F1 key at least twenty times. For each keystroke, it should print whether the Shift key modifier was present or not. The expected behavior is that the printout should display "Shift = true" for each keystroke. But after some number of iterations the printout starts to incorrectly display "shift = false" on some platforms.

### ***Analyze Result***

WIN95	Worked fine.
WIN98	Worked fine.
WINNT	Behaved as described.
LINUX	Worked fine.
SOLARIS	Worked fine.
MAC	Worked fine.

## **9. ScrollPaneTest**

**Type**           AWT

**Description**   Single click inside a scrollpane generates different numbers of events on different platforms. The test code brings up a window that has a scrollpane and four buttons in it. The scrollpane is created using the argument SCROLLBARS\_NEVER. The buttons are used to move the scrollbar up, down, left and right inside the scroll pane. A single click on any of those four buttons should generate a single event notification. For each of the events, a " Painting " message gets printed on the console window.



### ***Result***

WIN95	A single "Painting" message got displayed from a single click.
WIN98	A single "Painting" message got displayed from a single click.
WINNT	Double "Painting" messages appeared from a single click.
LINUX	Double "Painting" messages appeared from a single click.
SOLARIS	Double "Painting" messages appeared from a single click.
MAC	A single "Painting" message got displayed from a single click

### **10. SetSizeBug**

***Type***            AWT

***Description***    The SetSize(int, int ) method extends a frame from its normal size when a menu bar is attached to it. The test code displayed a frame with a button named "do". When start as "SetSizeBug 200 200", the application sets the size of the frame to 200x200. When the menubar is present, and the button named "do" is pressed it displays the size of the frame as 200x219 -- 19 pixels more.

### ***Analyze Result***

WIN95	Behaved as described.
WIN98	Behaved as described.
WINNT	Behaved as described.
LINUX	Worked fine.
SOLARIS	Worked fine.
MAC	Worked fine.

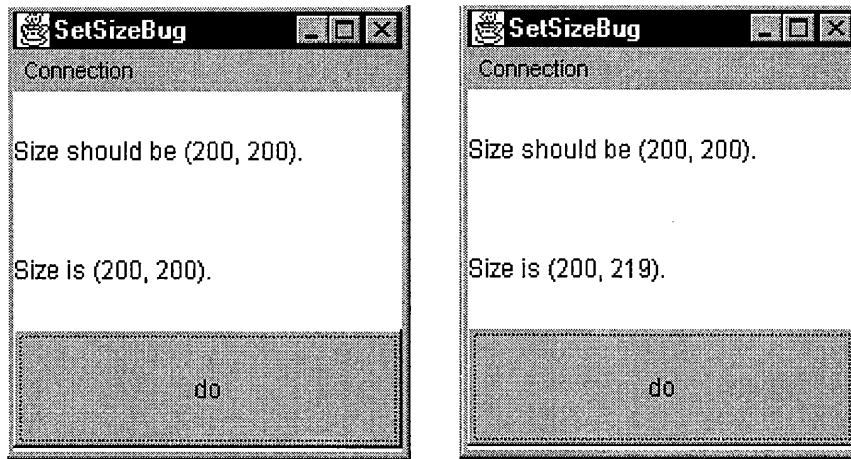


Fig. 25. Snapshot of SetSizeBug.java program under Win95 platform  
(wrong output).

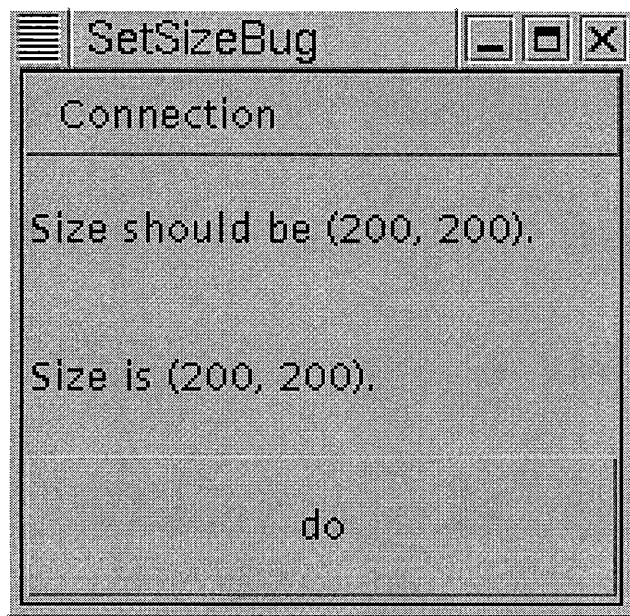


Fig. 26. Snapshot of SetSizeBug.java program under Linux platform  
(correct output).

## 11 Test2

**Type** AWT

**Description** The minimum size of a dialog box is often incorrect on some platforms. The test code shows a frame with a button named "A". When the button "A" is pushed a modal dialog box with another button named "B" gets created. When "B" is pushed, it hides itself. Now pushing "A" should show dialog "B" again by packing it back to the original size. But in the second case the "B" button in the dialog does not resize correctly on some platforms.

**Analyze** The problem appears to be in the preferred size area. Either the minimum size should be based on the minimum physical window size, or the components should be sized to the actual size of the container. Otherwise it produces incorrect sized components.

### **Result**

WIN95 Behaved as described.

WIN98 Behaved as described.

WINNT Behaved as described.

LINUX Worked fine

SOLARIS Worked fine.

MAC Worked fine.

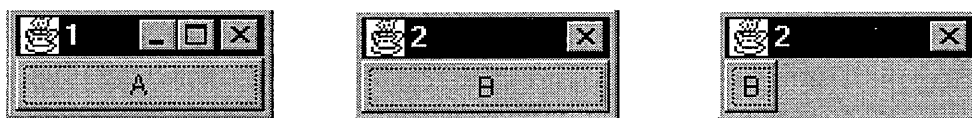


Fig. 27. Snapshot of test2.java program under Win95 platform (wrong output).

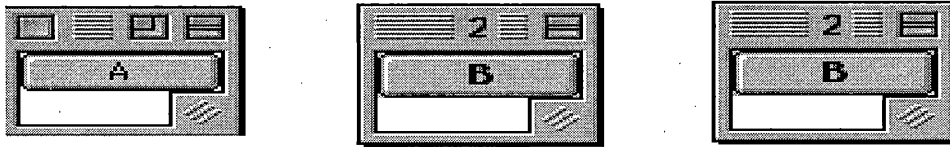


Fig. 28. Snapshot of test2.java program under Macintosh platform  
(correct output).

## 12 TestCase

**Type** AWT

**Description** The `ScrollPane.getInsets()` method does not return right inset value on some platforms. This test code brings up a window with two scrollpanes. If the `ScrollPane`'s size or the `ScrollPane`'s child component's size is changed, the `java.awt.ScrollPane` class's `getInsets()` method returns insets for previous state of the `ScrollPane` and the `ScrollPane`'s child component.

### *Analyze Result*

WIN95	Behaved as described.
WIN98	Behaved as described.
WINNT	Behaved as described.
LINUX	Worked fine.
SOLARIS	Worked fine.
MAC	Worked fine.

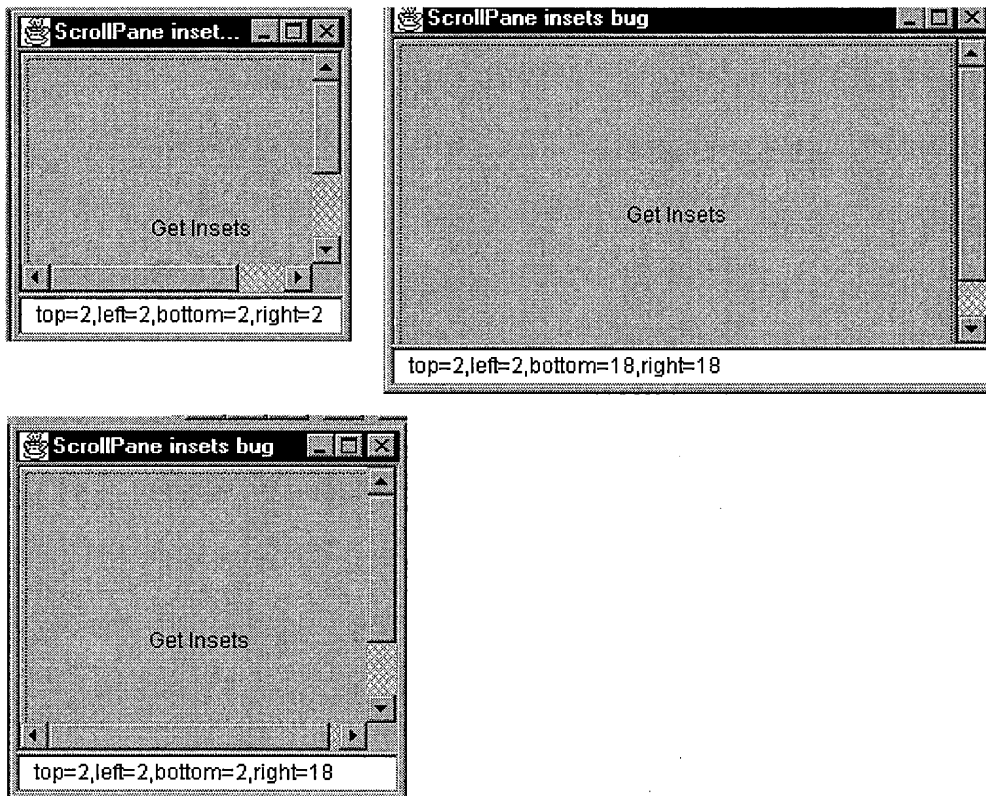


Fig. 29. Snapshot of TestCase.java program under Win95 platform  
(wrong output).

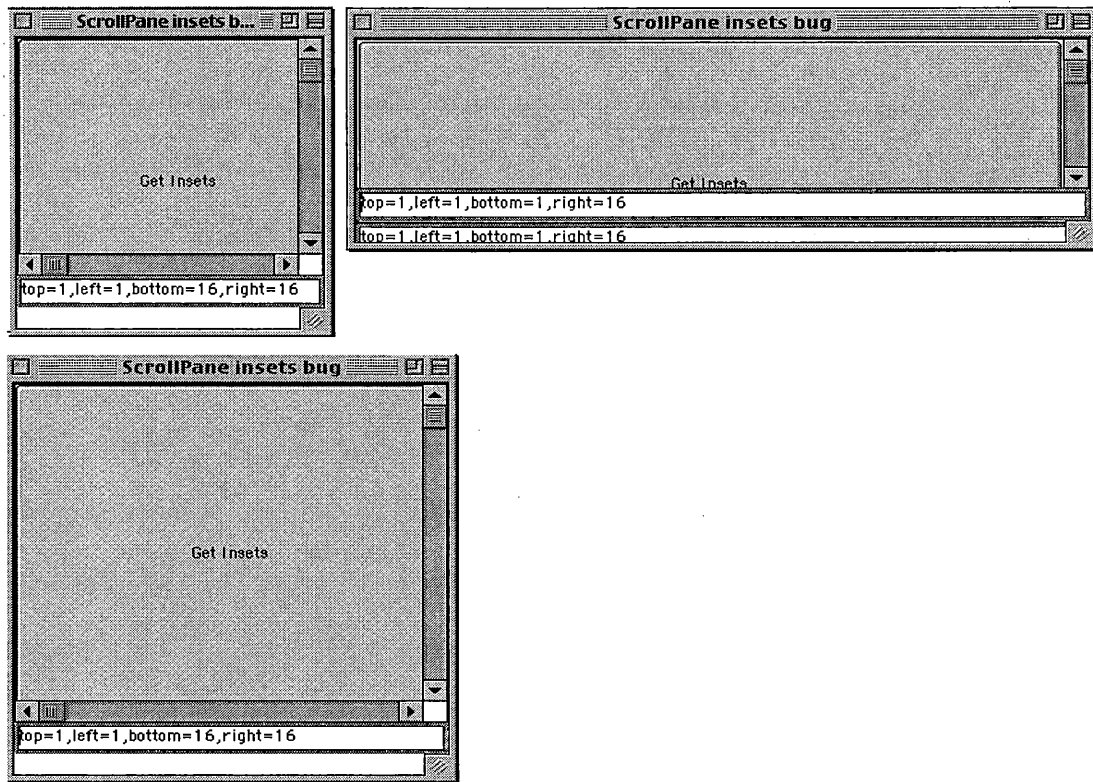


Fig. 30. Snapshot of TestCase.java program under Macintosh platform  
(correct output).

### 13 TestKeyListener

**Type** AWT

**Description** In some platforms "Ctrl" Key generates multiple KeyPressed events whereas in others it generates a single KeyPressed event. The test program brings up a window. When "Ctrl" key is pressed, it keeps on printing the generated events as print messages. It also prints a message when "Ctrl" key is released.

### **Analyze Result**

WIN95	Received multiple key pressed events when "Ctrl" key pressed and held down.
WIN98	Received multiple key pressed events when "Ctrl" key pressed and held down.
WINNT	Received multiple key pressed events when "Ctrl" key pressed and held down.
LINUX	Received a single key pressed event when "Ctrl" key is pressed and held down.
SOLARIS	Received a single key pressed event when "Ctrl" key is pressed and held down.
MAC	Did not generate any key press event when "Ctrl" key is pressed and held down.

## **14 Testpopup**

**Type** AWT

**Description** On some platforms labels, as MenuItems of a PopupMenu are not displayed when added to a list. The test program brings up a window with a popup menu with a list of menu items. In some platforms the labels do not get displayed.

### **Analyze Result**

WIN95	Empty popup menu was displayed.
WIN98	Empty popup menu was displayed.

WINNT	Empty popup menu was displayed.
LINUX	Labels were displayed inside popup menu.
SOLARIS	Labels were displayed inside popup menu.
MAC	Labels were displayed inside popup menu.

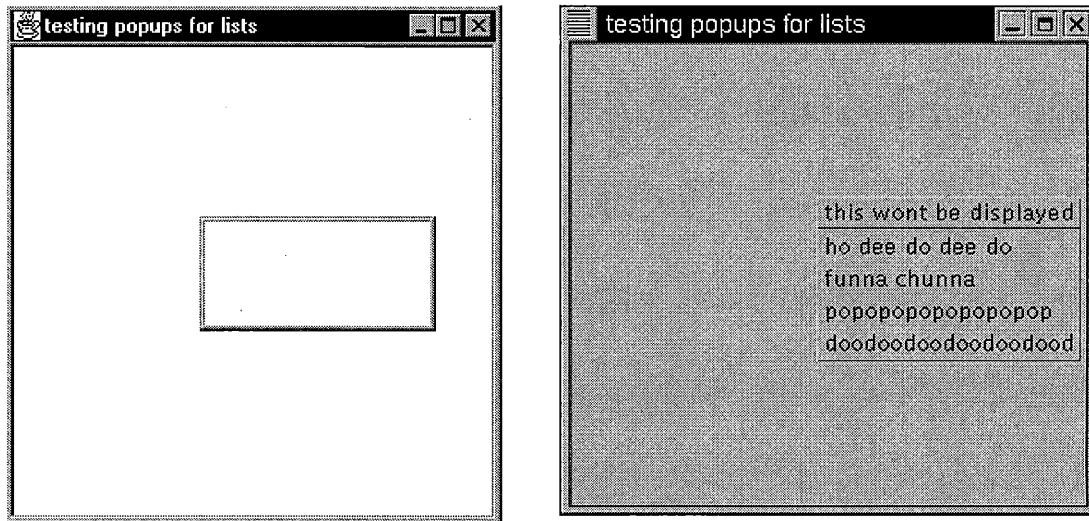


Fig. 31. Snapshot of testPopup.java program under Win95  
(left - wrong output) and Linux (right - correct output) platforms.

## 15 Testscrollbar

**Type** AWT

**Description** The bubble on the AWT scrollbar continuously blinks on some platforms. The behavior is not consistent across platforms.

### Analyze Result

WIN95	Scrollbar button consistently blinked.
WIN98	Scrollbar button consistently blinked.
WINNT	Scrollbar button consistently blinked.



LINUX	Scrollbar button did not blink.
SOLARIS	Scrollbar button did not blink.
MAC	Scrollbar button did not blink.

## 16 Testtextfield

**Type** AWT

**Description** The setBackground(Color) method does not function correctly on TextField on some platforms. The test program uses lightgray as background color that does not turn out right on all platforms.

**Analyze** Calling TextField.setEditable(false) prior to setting the background color does allow to set the background to lightGray (or orange, or other colors which use a bit combination other than all off or all on (or gray)).

### **Result:**

WIN95	Text field came up as white instead of lightgray when setEditable(false) is not used.
WIN98	Text field came up as white instead of lightgray when setEditable(false) is not used.
WINNT	Text field came up as white instead of lightgray when setEditable(false) is not used.
LINUX	Text field came up as white instead of lightgray when setEditable(false) is not used.
SOLARIS	Text field came up as white instead of lightgray when setEditable(false) is not used.

MAC           Text field came up as lightgray without using setEditable(false).

## **17   TFBehavior**

*Type*           AWT

*Description*   The TextField.select(int) method leaves the caretPosition in different position depending on operating systems, and provides wrong caret position on some platforms. The test program brings up a window with a text field in it. There is a button in the window which when pressed highlights a text displayed on the window from the tenth position onward and prints the caret position.

*Analyze*       On some platforms if the TextField.select(int) is called before the getCaretPosition() method, it returns the index of the end of the selection. However on others, getCaretPosition() returns the index of the beginning of the selection. This is in direct contradiction to what is seen on the screen; the insert cursor can be seen flashing at the end of the selection rather than the beginning.

### ***Result***

WIN95           [a] Sets the caret at the front after something was typed and "Enter" key was pressed.

                  [b] When the button was pressed, it sets the caret at the end but returned the index number of the beginning of the highlighted text as the caret position.

WIN98           [a] Sets the cursor at the front after something was typed and "Enter" key was pressed.

[b] When the button was pressed, it sets the caret at the end but returned the index number of the beginning of the highlighted text as the caret position.

WINNT

[a] Sets the cursor at the front after something was typed and "Enter" key was pressed.

[b] When the button was pressed, it sets the caret at the end but returned the index number of the beginning of the highlighted text as the caret position.

LINUX

[a] Sets the caret at the end after something was typed and "Enter" key was pressed.

[b] When the button was pressed, it sets the caret at the end and returned the right caret position.

SOLARIS

[a] Sets the caret at the end after something was typed and "Enter" key was pressed.

[b] When the button was pressed, it sets the caret at the end and returned the right caret position.

MAC

[a] Sets the cursor at the front after something was typed and "Enter" key was pressed.

[b] Did not show the caret. Only highlighted the region, and returned the front position as caret position when the button was pressed.

## 18    **TypeAhead**

**Type**            AWT

**Description**    While a new window is being shown or activated, key events do not go to a predictable place and are often lost. This means users cannot type ahead of the application - that is, they cannot enter input for windows that have not yet fully come up yet. This test program brings up a window that has a single button in it. The button gets activated when space bar is pressed, and it opens up another window with a text field. If the user presses a space bar and starts typing before the second window gets visible the typed characters do not appear in the text field of the second window, in some platforms.

**Analyze**        Probably the problem is that there is a gap between the time a window is shown or activated and the time at which focus gets set to a component in that Window. Any key events that come in during this gap will go to an undefined location. On windows, the focus will stay in the old window, then shift to the frame of the new window, then shift to the component which should get focus in the new window. So type-ahead keys will go to either the component that had focus in the old window or the frame of the new window. Neither of these is what the user expects.

### **Result**

WIN95	Behaved as described.
WIN98	Behaved as described.
WINNT	Behaved as described.
LINUX	Behaved as described.

SOLARIS      Behaved as described.

MAC            Worked fine.

## **19      WinDiaFocus**

*Type*            AWT

*Description*    On some platforms, windows and dialogs often do not get focus. The test program creates a dialog, a window and a frame window to verify this issue.

### ***Analyze Result***

WIN95          Focus gained and lost event occurred from all three components.

WIN98          Focus gained and lost event occurred from all three components.

WINNT          Focus gained and lost event occurred from all three components.

LINUX          Focus gained and lost event occurred only from frame window.

SOLARIS        Focus gained and lost event occurred only from frame window.

MAC            Focus gained and lost event occurred only from frame window.

## **20      WindowTest**

*Type*            AWT

*Description*    In some platforms non-resizable frames often become a transparent window; and clicking on it passed through to any object behind the window.

### ***Analyze Result***

WIN95          Behaved as described.

WIN98          Behaved as described.

WINNT	Behaved as described.
LINUX	Worked fine.
SOLARIS	Worked fine.
MAC	Worked fine.

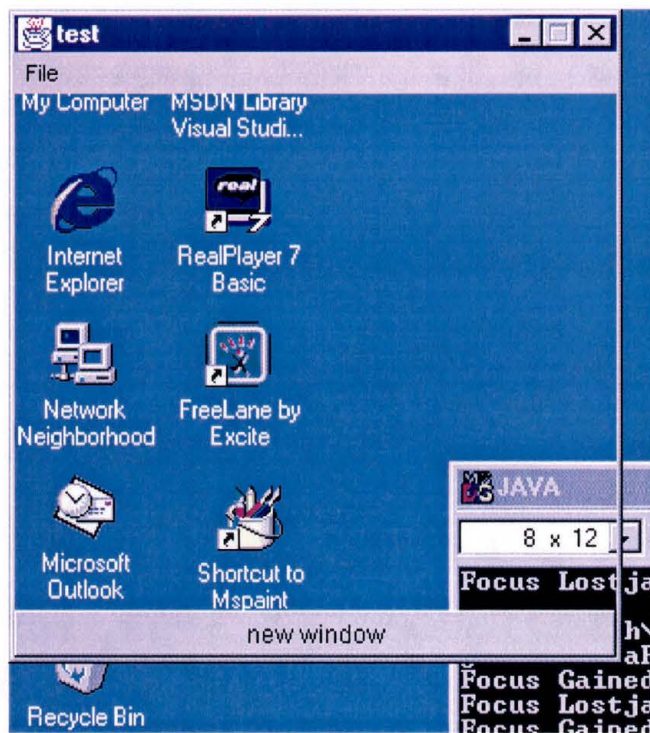


Fig. 32. Snapshot of WindowTest.java program under Win95 platform

(wrong output).

## 21 WndTest

**Type** AWT

**Description** If the setLocation(int, int) method moves a frame or a window, the frames and the windows inside the parent frame or window redraw. The test program creates a canvas window with multiple child windows inside it with solid

circles of different colors. Pressing the "Enter" key causes the parent window to shift its position. With every shift the child windows repaint themselves along with the parent window.

### ***Analyze Result***

WIN95	Behaved as described.
WIN98	Behaved as described.
WINNT	Behaved as described.
LINUX	Did not redraw.
SOLARIS	Did not redraw.
MAC	Did not redraw.

### **IO Based Programs**

#### **22      Factors42**

***Type***            IO

***Description***    BigInteger's modPow(BigInteger, BigInteger) method failed on most platforms when an input that is powers of 2 is used. This test program prompts the user to input a number and returns a BigInteger whose value is (this \*\* exponent) mod m. (If exponent == 1, the returned value is (this mod m). If exponent < 0, the returned value is the modular multiplicative inverse of (this \*\* -exponent).). The program throws an ArithmeticException if m <= 0.

### ***Analyze Result***

WIN95	Crashed at runtime when 4 was inputted.
WIN98	Crashed at runtime when 4 was inputted.

WINNT	Crashed at runtime when 4 was inputted.
LINUX	Crashed at runtime when 4 was inputted. It provided a full thread dump, monitor cache dump, and registered monitor dump before termination.
SOLARIS	Crashed at runtime when 4 was inputted.
MAC	Worked fine.

```

4
2
2

D:\Research\2ndSemester\FinalTestResult\Run\CompiledUnderWin95\Code\Factors42>javac Factors42
javac: invalid argument: Factors42

D:\Research\2ndSemester\FinalTestResult\Run\CompiledUnderWin95\Code\Factors42>javac Factors42.java

D:\Research\2ndSemester\FinalTestResult\Run\CompiledUnderWin95\Code\Factors42>java Factors42
25
5
5
4
Assertion failed: bits > 1, file .....\src\share\java\math\bn-1.1\lbn16.c, line 3063
abnormal program termination
D:\Research\2ndSemester\FinalTestResult\Run\CompiledUnderWin95\Code\Factors42>

```

Fig. 33. Snapshot of Factors42.java program under Win95 platform

(wrong output)

## 23 Jitbug

*Type* IO

*Description* On most platforms this program generates the following "JIT error" message.



"A nonfatal internal JIT (3.00.055(x)) error 'BinaryNonCommutative' has occurred."

Purpose of the test was to verify the problem in all platforms.

**Analyze Result**

WIN95	JIT error message occurred although the program presented the correct output.
WIN98	JIT error message occurred although the program presented the correct output.
WINNT	JIT error message occurred although the program presented the correct output.
LINUX	Worked fine without JIT error.
SOLARIS	JIT error message occurred although the program presented the correct output.
MAC	JIT error message occurred although the program presented the correct output.

**24 ColorTest**

**Type** IO

**Description** The system colors are incorrectly initialized under some platforms.

The purpose of this test was to verify the behavior on other platforms.

**Analyze** In some platforms, high byte is initialized to 0x00 while spec says: getRGB() method "Gets the "current" RGB value representing the symbolic color. (Bits 24-31 are 0xff, 16-23 are red, 8-15 are green, 0-7 are blue)."

## **Result**

WIN95	All system colors are initialized as 0.
WIN98	All system colors are initialized as 0.
WINNT	All system colors are initialized as 0.
LINUX	The program hanged without printing any value.
SOLARIS	The program hanged without printing any value.
MAC	Macintosh printed fewer attributes; and all of those are wrongly initialized as 0.



Fig. 34. Snapshot of ColorTest.java program in Win95 platform

(wrong output).

The following test programs are described and analyzed in the body of the research.

**25     Calculator**

*Type*            AWT / *Swing*

**26     EventDemo**

*Type*            AWT

**27     FontPicker**

*Type*            AWT/*Swing*

**28     PlafTest**

*Type*            AWT

**29     AltTest**

*Type*            AWT/*Swing*

**30     ButtonTest**

*Type*            AWT

**31     FrameSize**

*Type*            AWT

**32    GetSourceBug**

*Type*            AWT

**33    Labelbug**

*Type*            AWT/Swing

**34    Example5**

*Type*            IO

**35    ClassLoaderSyncProblem**

*Type*            IO

**36    MsgLog**

*Type*            IO

**37    ExecTest**

*Type*            IO

**38    FileCopy**

*Type*            IO

**39    FindDirectories**

*Type*            IO

**40    Readdata**

*Type*            IO

**41    Hello2**

*Type*            IO

**42    LineTerminationTest**

*Type*            IO

## APPENDIX C

Table 11. List of vendors that support Java portal work

Operating System	Vendors
AIX	IBM
DG/UX 4.2	Data General Corporation
DYNIX/ptx 4.4.2 forward	Sequent Computer Systems
HP-UX	Hewlett Packard
IRIX	Silicon Graphics
Linux	Blackdown.org
MacOS	Apple Computer
Netware	Novell
OS/2	IBM
Os/390	IBM
OS/400	IBM
OpenVMS	Compaq Computer Corporation
Tru64 UNIX	Compaq Computer Corporation
SCO	SCO
UnixWare	SCO

## WORKS CITED

- Cramer, T., Friedman, R., Miller, T., Seberger, D., Wilson, R., Wolczko, M.  
"Compiling Java Just in Time," *IEEE Micro*, Vol. 17, No. 3, (May/June 1997).
- Edmunds, A. R. *The Prentice-Hall Standard Glossary of Computer Terminology*.  
New Jersey: Simon & Schuster, 1984.
- Edwards, M. "Let's Talk About Java Portability", Microsoft Technologies for  
Java, (May 1997).
- Rabinowitz H., Schaap C., *Portable C*, Prentice Hall, Englewood Cliffs, New  
Jersey, 1989.
- Horsmann, C. S. Cornell, G. *Core Java Volume I-Fundamentals*,  
California: Sun Microsystems Press, 1999.
- Hudgins-Bonafield, C. Java's Future, *Information Week*,  
URL : <http://iweek.com/705/05iujav.htm>, 1998.
- Linden, V. P. *Not Just Java A Technology Briefing*, 2d ed. California: Sun  
Microsystems Press, 1999.
- Morrey, B. "Java in the enterprise," *InfoWorld*, Vol. 19, issue 37 (15 September  
1997).
- Nilsen, K. "Issues in the Design and Implementation of Real-Time Java," *Java  
Developer's Journal*, Vol. 1, Issue 1 (1997).
- Rofrano, J. "Java Portability by Design - An effective way to encapsulate the  
system differences," *Dr. Dobb's Journal*, June 1999.
- Scott, B. "Anonymous Deployment vs Portability," *Java World*, Vol. 4, Issue 10  
(October 1999).
- Sun Microsystems, Inc. *100% Pure Java Cookbook for Developers – Rules and  
Hints for Maximizing Portability*, 1999, URL: <http://java.sun.com/100percent/>
- Venners, B. *Inside The Java Virtual Machine*, 2d ed. New York(???): McGraw-  
Hill Professional Publishing, 1999.

Warren, J.C., "Software Portability. A Summary of Related Concepts and Survey of Problems and Approaches", Technical Note No. 48, Stanford University, Stanford, California (September 1974).



## VITA

Tanvir Rahman was born in Dhaka, Bangladesh, on February 12, 1972, the son of Lutfur Rahman and Laila Rahman. He completed his high school in Dhaka, Bangladesh, and received his Bachelor of Science in Electrical Engineering in December 1995 from the University of Texas at Austin. Since January 1996, he has been working at VTEL Corporation as a software engineer. He is currently pursuing for his Master's degree in Software engineering at the Southwest Texas State University, San Marcos, Texas.

Permanent Address:           1305 Merchants Tale Lane  
Austin, Texas 78748

This thesis was typed by Tanvir Rahman.