No. 2013-1021 & No. 2013-1022

IN THE

UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

ORACLE AMERICA, INC.,

Plaintiff-Appellant,

v.

GOOGLE INC.,

Defendant-Cross Appellant.

ON APPEAL FROM THE UNITED STATES DISTRICT COURT FOR THE NORTHERN DISTRICT OF CALIFORNIA, CASE NO. 10-CV-3561 HON. WILLIAM H. ALSUP

CORRECTED BRIEF OF SCOTT MCNEALY AND BRIAN SUTPHIN AS *AMICI CURIAE* IN SUPPORT OF REVERSAL

STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@
CliffordChance.com

Counsel of Record
Counsel for Amici Curiae

February 22, 2013

CERTIFICATE OF INTEREST

Pursuant to Federal Rules of Appellate Procedure 26.1 and 47.4, Counsel for *Amici Curiae* certifies the following:

1. The full name of every party or Amici Curiae we represent is: Scott McNealy and Brian Sutphin

2. The name of the real party in interest represented by us is: None.

3. All parent corporations and any publicly held companies that own 10 percent or more of the stock of the party or amicus curiae represented by us are: None.

4. The following law firm and partners or associates have appeared or are expected to appear before this Court on behalf of the Amici Curiae are:

STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@
CliffordChance.com
Counsel of Record

Dated: February 22, 2013

By: /s/ Steven T. Cottreau
STEVEN T. COTTREAU

TABLE OF CONTENTS

Page

TABLE OF AUTHORITIES

INTEREST OF AMICI CURIAE

Amici Curiae Scott McNealy and Brian Sutphin ("Sun Executives") are former executives of Sun Microsystems, Inc. ("Sun"), who were integrally involved in the development of the Java platform. Sun, which was founded in 1982, developed the world's most innovative products and services, which have been used to power the world's key computing systems. Through its commitment to shared innovation, community development, and open source leadership, Sun quickly became a leader in the sale of computer workstations. In the 1990s, Sun developed Java, which is an object-oriented, platform-independent, multithreaded programming environment. Java revolutionized computer programming and quickly became the foundation for the World Wide Web and numerous computing and networking devices.

Amicus curiae Scott McNealy co-founded Sun and was the Chairman of its Board of Directors from 1984 to 2010, President from December 1984 to April 1999 and from July 2002 to April 2004, and Chief Executive Officer from December 1984 to April 2006. In these roles, Mr. McNealy worked to make Sun an innovative leader in the information-technology industry.

Amicus curiae Brian Sutphin joined Sun in 1994 and from 2004 through 2010 was Sun's Executive Vice President of Corporate Development and Alliances. Mr. Sutphin's responsibilities included mergers and acquisitions, creating

corporate-level alliances with key global technology companies and establishing policies and requirements of Sun's inbound technology licensing.

In 2010, Oracle Corp. acquired Sun and the Sun Executives moved on to other projects. While the Sun Executives do not have any current involvement with Oracle or Java, they share a vital interest in protecting Java's creative legacy.

Because of their close involvement with the creation of Java, the Sun Executives are compelled to submit this brief to make clear that Java was the result of the exceptional creative efforts by a team of developers and to safeguard Sun's legacy. The Sun Executives urge this Court to safeguard those individuals' legacy, which the opinion below has placed in doubt.[1]

[1] Pursuant to Federal Rule of Appellate Procedure 29(a), both parties have consented to the filing of this brief. Moreover, no party to this case or its counsel authored this brief in whole or in part, or no person other than amici and their counsel made a monetary contribution to its preparation or submission. *See* Fed. R. App. P. 29(c)(5).

ARGUMENT

The Sun Executives are concerned that the district court did not fully appreciate the creativity involved in developing the Java Platform or the countless choices that Sun made in developing it. First, the district court's opinion assumes that each of the 7,000 lines of source code that Google copied could only be written in a small number of ways. In creating Java, however, the Sun development team could have used an unlimited number of syntax permutations to develop the same functionality that is generated by the Java Packages. Second, the district court's opinion refused to grant copyright protection to the Java Packages' highly creative organization, which elegantly interrelates classes and methods. In denying copyright protection to the copied elements of Java, the district court has upset the expectations of those that created the Java platform, who thought that the source code that they painstakingly developed would receive the same copyright protections afforded to any other source code (or literary work). Third, the district court's opinion undermines the financial investment that Sun made in developing Java and the licensing framework that the Sun Executives created to enforce their unique "Write Once, Run Anywhere" philosophy. Had Sun known a priori that the result of investment of millions of dollars and years of development time would not receive copyright protection, it never would have invested as heavily in Java.

I. OVERVIEW OF JAVA FRAMEWORK

A. Brief History Of Java

Prior to the release of the Java platform, the dominant programming languages permitted developers to use some common rules and vocabularies in writing programs for different types of computer systems and devices. But, because each type of computer systems or device was unique, a program written in one of these languages had to be written in a manner that was specific to a particular system or device. Thus, although a programmer could write programs for multiple systems or devices in the same language, the program itself would have to be re-written (or "ported") in a manner that was specific to each particular platform or device. For example, an application written in the programming language C and designed for the Microsoft Windows system would not work on an Apple computer or on any other non-Windows device.

Porting for multiple systems, however, was often prohibitively expensive and massively inefficient. To port, a developer would have to assign a team of programmers to take a completed program and change its programming source code so that it would work on a new platform. Even if a developer was willing to incur the expense, the porting process took valuable time and thus slowed the process of making software available for new platforms and devices. As a result,

developers often chose to write software for systems with the largest number of users. Early efforts to develop a cross-system platform failed.

Innovators at Sun realized they needed to start "really [bearing] down and [starting] to help customers solve the problems they were having in migrating away from mainframes."[2] A team of Sun computer engineers led by James Gosling worked feverishly on a computer programming platform with the intent that it would revolutionize how people would program. They developed Java, a breakthrough platform that permitted developers to "**Write Once, Run Anywhere**." Software developers could use Java to write an application once using Java and have it run on a variety of different computing systems and devices.

Java's "Write Once, Run Anywhere" promise was an inspirational creative breakthrough in software development. Unlike programs written on predecessor programming platforms, a program written and developed once with the Java development platform would work on a very large selection of systems and devices, thus freeing developers from the high costs of porting their applications.[3]

Central to Java's success was its inclusion of a collection of creatively pre-written programs bundled into "packages" (also known as APIs or Application

---

[2] Lee Gomes, *Made in the shade; 'Java' stirs up renewed interest in Sun Micro*, The Dallas Morning News, Dec. 18, 1995.

[3] David Bank, *The Java Saga*, Wired Magazine, Dec. 1995, *available at* http://www.wired.com/wired/archive/3.12/java.saga.html?topic=&topic_set=.

Programming Interfaces) that allowed programmers to code quickly and efficiently. The choice of what packages to create, the creativity of the naming of those packages, and creativity in the organization of those packages are at issue in this case.

Due to its elegant and robust design, the Java platform has proven an enormous success. The Java platform has evolved and thrived for over 15 years while competing against myriad other notable development platforms. At present, more than 9 million developers worldwide create software with Java, making Java the most popular programming platform.[4] In fact, Java gained the top position in programming platforms more than 10 years ago, in 2002.[5] Java's success not only advanced "Write One, Run Anywhere," but it was the forefront of the World Wide Web revolution of the 1990s, "bring[ing] the Internet to a new level of experience for all users."[6] Near the time of Java's initial release, Netscape Communications said it "[had] never seen interest in a programming language grow so fast as it has

---

[4] Learn About Java Technology, Java.com, http://www.java.com/en/about/ (last visited Feb. 16, 2013).

[5] TIOBE Programming Community Index for February 2013, TIOBE Software: The Coding Standards Company, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html?date=jan2013 (last visited Feb. 16, 2013).

[6] Elizabeth Corcoran, *Java Jumps Into the 'Net'; Proponents Say New Software Language Could Herald Computing Revolution*, The Washington Post, Dec. 10, 1995.

with the phenomena of Java."[7] The Washington Post had noted before its release that "[to] many technologists, describing Java as a programming language is like dismissing iron ore as just another type of rock—rather than a material that can be used to create a new age of tools. What's exciting about Java isn't what it is, but what people hope to do with it."[8] Realizing that desktop computers were a sliver of the potential systems that Java could innovate, Java's creators also released specialized platforms to fully exploit what were, at the time, unconventional. For servers and mainframes, an Enterprise Edition was released and Sun pioneered mobile phone development with its Micro Edition. When Google made Android available for free, Oracle was effectively competing with a free version of its own program.

B. How Java Works

The Java platform relies upon five primary elements: (i) a Java programming language with which developers can write applications; (ii) a core set of programs (interchangeably called the **Java Packages, Java APIs, or Java Class Library**) which developers could use to speed the creation of new applications; (iii) a Java compiler, which translates the code written by the developer into Java bytecode;

---

[7] Patrick McKenna, *Netscape Creates Java Conference*, Newsbytes, Dec. 20, 1995.

[8] Corcoran, *supra* note 6.

(iv) a Java virtual machine ("JVM"), which translates Java bytecode into instructions comprehensible to the underlying computing platform or device; and (v) a Java development kit ("JDK"), a collection of programming tools released by Oracle. If the Java Class Library and a JVM are present on a system or device, the system is said to carry a "Java runtime environment" and that system then can run applications written in Java. Of these four primary elements of Java, only Google's copying of the Java Packages (or APIs) is at issue in this case.

1. ***Java Language***. The Java language constitutes the "bare bones" of the Java framework. The language provides the syntax, grammar, and vocabulary of the language to permit a software developer to write programs in Java code that will run in a Java runtime environment.

2. ***Java Packages***. Java provides an extensive set of packages organized into a library—those packages are at issue in this case. The Java Packages are an extensive set of ready-to-use programs that can serve as helpful "building blocks" for Java developers. Sun developed these pre-built packages to allow programmers to accomplish programming tasks ranging from the simple (such as basic math functions) to the complex (such as providing computer security and network access functions). In other words, Java developers need not "reinvent the wheel" for many desired programming tasks that were commonly used by a wide

variety of computer programs. Rather, the creators of Java included these functions in the Java Packages.

Although there are many intricacies to how the Java Packages interact with one another and within themselves, a package is generally subdivided into classes or interfaces, which are further subdivided into methods. Each method contains the discrete programming functions utilized by developers. For example, the java.net package provides 40 classes and interfaces to implement networking applications (*i.e.*, connecting to the internet and other computer networks). It contains 440 methods that range from determining if the computer or device is connected to a locally networked computer, to retrieving the IP address (i.e., the address box) of another computer connected to the Internet.

3. ***Java Compiler.*** The Java compiler reads and interprets the source code written by a developer, including its declaring code to the Java Packages, and generates a more compact Java byte-code. This byte-code, which cannot be read directly by most Java programmers, is what is distributed to end-users to run on different computing platforms.

4. ***Java Virtual Machines (JVM).*** JVMs are the final link to Java's "run anywhere" platform. Installed on a user's computer or device, the JVM is a piece of software that translates the Java byte-code to enable it to run on each particular computer or device. In short, to run Java programs on a particular computer

platform, the JVM must be customized for that platform. Once a JVM exists for a platform, all computers and devices using that type of platform can run programs written in Java. Today, JVMs have been developed for almost every computer operating system (including Microsoft Windows, Apple MacOS X, Linux variants, and UNIX variants) and an expanding number of mobile devices (such as Blackberry).

5. ***Java Development Kit (JDK).*** The JDK is the culmination of all the elements of the Java platform. Released by Java's creators, the JDK provides all the necessary programs and tools to develop and test their Java applications. The JDK includes the Java Packages, the Java Compiler and the JVM. By providing all the pieces of the Java platform, the widespread distribution of the JDK was fundamental to promoting Java's "Write Once, Run Anywhere" vision.

II. JAVA IS CREATIVE AND INNOVATIVE

A. Designing An Elegant Set Of Java Packages Was Central To Java's Success

Java's success rested in large part upon its elegant and creative set of packages that Sun designed and developed. As noted above, these packages provide a lengthy and creative set of pre-existing programs that made it much easier for Java programmers to quickly write programs and intuitively grasp and learn the Java platform. Indeed, Java's well-crafted packages have been an

important contributing factor to the success that Java has enjoyed since its inception.

Merely offering a programming platform in which developers could build cross-system applications would not have been enough to get developers to use it. Instead, the platform needed an elegantly designed and arranged package library that would resonate with programmers' sensibilities and that would be intuitive to their thinking. A creatively developed, labeled and organized set of packages was essential to the spread of a new computer platform because it makes the platform easier to learn and to teach. Although the conventions and ordering of the packages may read like gibberish to non-programmers, an elegantly designed set of packages is a creative exercise that is apparent to everyday professional programmers—in the way that an English professor can recognize a unique arrangement of words as great poetry or a renowned architect can see lines in a blueprint as the next iconic silhouette in a city skyline. The function is not the key—instead, success is found in the artful selection of elements: in Java, this includes the elegant naming and creative arrangement of the Java Packages.

As a result, Sun spent an enormous amount of time and effort creating an original hierarchy of packages organized into a distinctive library and unique classes. To attract developers to invest the time to learn the platform, the selection and then naming and organization of the packages had to be easy to understand, to

memorize, and to master. Indeed, because Java's packages are so appealing to Java developers, a community devoted to the development of new packages and the evolution of existing packages was established. This process of package design took many years to develop and the development continues to this day. The result has been a monumental creative success.

The development of the Java Packages is not a casual effort that took a few weeks to perfect. Sun/Oracle took years to refine the Java Packages. The process to cultivate a package can take over a year from start to finish. There are four major steps: (i) Initiation; (ii) Draft Releases; (iii) Final Release; and (iv) Maintenance. During the initiation phase, a new specification is requested by an accredited community member, which in turn must be approved by an executive committee established by Oracle. Experts then draft the specification and submit it to the Oracle's Program Management Office, where it is posted online for public feedback. The specification is revised, utilizing the public feedback, and reposted to the public repeatedly during a set period. Once that period is over, a ballot is taken, and if passed, the final release is published. The implementation of the final release must pass a battery of tests, most importantly compatibility with the

existing Java Packages. At that point, a specification becomes an official component of the Java Packages.[9]

B. Google's Employees and Other Individuals Have Recognized the Elegance of Java's design

Google's own Principal Engineer (who previously worked as Senior Sun Engineer on the Java project), Joshua Bloch, has stressed the importance of the creative process that leads to creation of an elegant package library. He has explained, "APIs can be among a company's greatest assets."[10] He stresses that integral to the process of package design is naming.

> The names that you come up with for classes and for methods sort of—they're talking to you …. They should come out nicely. They should work nicely together. You know, good names can drive development.'[11]

As a result, he notes that "Names matter a lot." He concludes that if a programmer creates elegant names, "then your code will kind of read like prose."[12] He has stressed the critical element of literary creativity in naming and organizing

---

[9] JCP Procedures - JCP 2.9: Process Document, Java Community Process, http://www.jcp.org/en/procedures/jcp2_9 (last visited Feb. 18, 2013).

[10] Joshua Bloch, *How to Design a Good API & Why it Matters*, Javapolis Conference, Nov. 21, 2006 *available at* http://www.infoq.com/presentations/effective-api-design.

[11] *Id.*

[12] *Id.*

packages: "API design is an art, not a science. Strive for beauty, and trust your gut."[13]

As a leading Java educator has explained, "What has impressed me most as I have come to understand Java is that somewhere in the mix of Sun's design objectives, it seems that there was a goal of reducing *complexity for the programmer*."[14] Indeed, countless programmers have noted the creative design of Java and its packages. As one programmer has put it,

> When I first began to program in Java, I loved the Java language a lot. I used to program in Pascal, Delphi, Visual Basic and C but Java was very different and elegant. In addition to its language structure and features, its API set was very special. With its beautiful and aesthetic design, programming in Java is a pleasure. I don't have this feeling when I program in other languages. To feel pleasure or pain is also valid when we use API sets. There are many API sets we use in any development cycle coming from different frameworks or libraries. API beauty depends on designer knowledge and design capability (say artistic skill).[15]

As a result of the creative selection, naming and organization of these packages, Java has become one of the most enduring programming platforms ever conceived due to its creative design and elegant organization.

---

[13] Joshua Bloch, *Bumper-Sticker API Design*, InfoQ, Sept. 22, 2008, http://www.infoq.com/articles/API-Design-Joshua-Bloch.

[14] Bruce Eckel, Thinking in Java 1 (4th ed. 2006).

[15] Ibrahim Levent, *Beautiful API Design*, DZone, Nov. 26, 2008, http://java.dzone.com/news/beautiful-api.

C. The Selection Naming And Organization Of Java's Packages (APIs) Are Unique And Creative

Java's package library design is unmistakably highly creative and the product of a rigorous design process. The development of the Java Packages goes far beyond mere function—they instead share many similarities with developing literature and architectural design. While a novel may have an object of stirring certain thoughts or emotion in a reader (a function), a novel or poem is ultimately a highly creative process that requires careful consideration to a variety of literary concerns, including structure, word choice, and pacing. Likewise, though a building has functional components like doors and windows, the overall arrangement of its elements is recognized as a highly creative medium of expression. Similarly, though a programming library contains functional components, the naming and organization of the functions—like the arrangement of architectural functions—is a highly challenging and creative process. The result is an intuitive, easy-to-comprehend, original scheme of Java packages.

The role of the Java Packages is to provide a library of shortcuts to allow the developers to quickly and efficiently create their own applications in Java. But, much like a haiku poem can express a particular thought, a package should also concisely relay the concepts and role of that program in an elegant and clever way—a way that avoids forcing the developer to resort to a manual to understand it. Good packages are intuitive and resonate with programmers. Consequently,

creative naming, organization, and classification of the packages have been fundamental to the success of the Java platform.

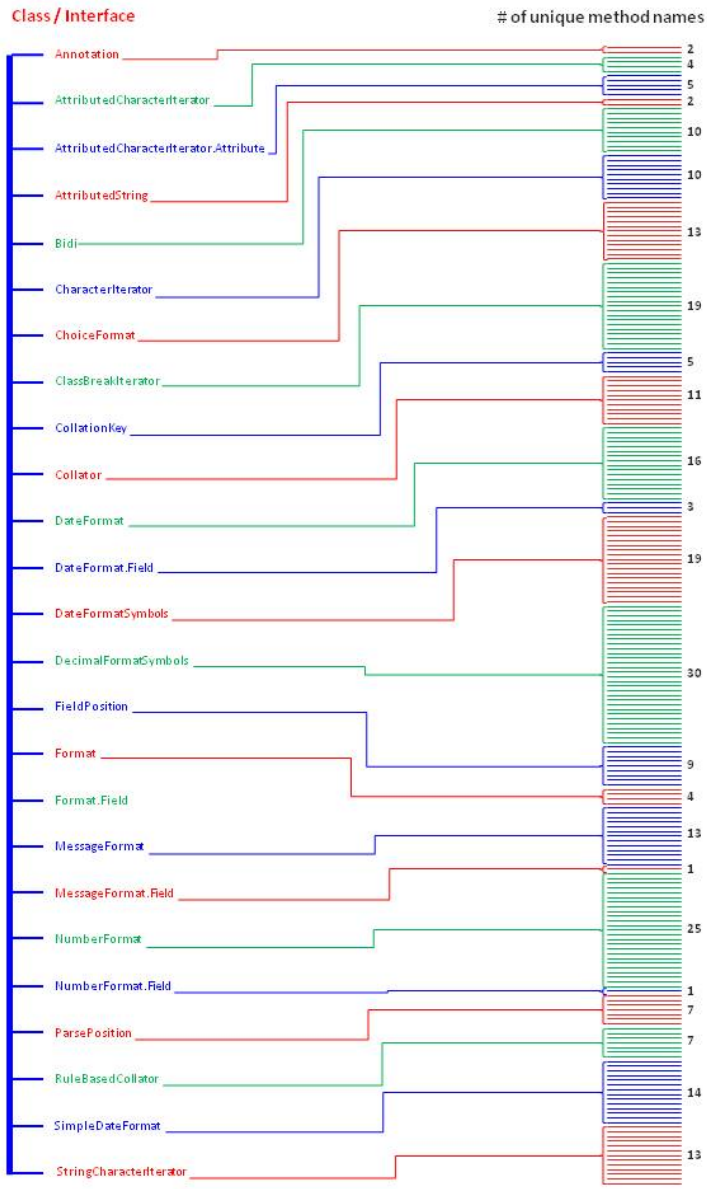
Java's architects had to decide what packages would be created, and in those packages, how many classes, and within those classes how many methods would ultimately be available to developers. Moreover, for every package, class, and method that made the cut, Java's creators then had to name each element, determine the types and order of their parameters, and determine how to arrange them. Over time, Java's developers evolved existing packages and added new ones. Like characters in a long running episodic television drama, the packages became increasingly interrelated and more sophisticated while new ones were introduced.

D. Designing a Java Package

An examination of the "java.text" package from Java Standard Edition 5.0 is illustrative of the myriad creative choices the Java package creators faced. The java.text package contains 25 classes, 2 interfaces, and hundreds methods to handle text, dates, numbers, and messages in a manner independent of natural human languages to allow the flexibility of adding localization for new localizations at

any time.[16] Figure A sets forth an overview of this Package's structure and Figure B below lists each of the methods within this Package.

Figure A- java.text Package



[16] java.text (Java 2 Platform SE 5.0), API Specification, http://docs.oracle.com/javase/1.5.0/docs/api/java/text/package-summary.html (last visited Feb. 18, 2013).

Java's creators initially needed to determine whether to include a java.text package in the first place. Just as a novel's author would need to determine the desirability of including a particular plot line, determining whether to include a package was a difficult creative choice. That, however, is just the start of thousands of creative choices that go into drafting a Java Package library. Java's architects then had to determine how long the package would be, what elements to include, and where to end the package. Moreover, the packages, classes, and methods had to have easy to grasp names that were creatively unique. Beyond the names, Java's developers had to make creatively appealing organizational choices. For example, organizing the java.text package alphabetically or chronologically were possibilities, but not creative choices that Java's creators made. The names of the classes and the way they were organized could not be so lengthy that it would be inefficient for programmers to use regularly, yet the labels still needed to creatively convey the purpose of the package. Moreover, Java's architects needed to consider how java.text would be interrelated with the broader Java Package library. Java.text is utilized by 8 other packages, including packages dedicated to fonts, user interfaces, and images.

A building architect must consider the interaction of the elements in a creative work to expresses an aesthetic perspective beyond function, so to did Java's architects carefully design the naming and arrangement of the elements

within the Java Packages to reflect their creative design philosophy. All of these choices are creative choices—not choices that in any way alter the functions of the packages available on the Java platform. For instance, the selection of names of a package's elements are not purely functional choices--they are creative ones. In Java, a developer setting the time zone in an application would first go into the "DateFormat" class of the java.text package and declare the "setTimeZone" method.[17] By just looking at their labels a developer will intuitively know that the DateFormat class can be used to format a date, and then use the setTimeZone method to set the actual time zone for that developer's application. But creators of competing computer programming environments can accomplish this same function in an unlimited number of different creative ways.

Indeed, a quick examination of other programming environments shows that creators of other development platforms provide the same functions with wholly different creative choices. For example, Apple's iOS platform devotes an entire class to set the time zone in an application-- the "NSTimeZone" class.[18] Unlike Java's placement of that package in the java.text package, Apple put it in

---

[17] DateFormat (Java 2 Platform SE 5.0), API Specification, http://docs.oracle.com/javase/1.5.0/docs/api/java/text/DateFormat.html (last visited Feb. 18, 2013).

[18] NSTimeZone Class Reference, Mac Developer Library, https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSTimeZone_Class/Reference/Reference.html (last visited Feb. 18, 2013).

its ”Foundation” framework. (A framework is Apple’s terminology for a structure conceptually similar to Java’s “package.”). Apple’s NSTimeZone class contains numerous methods to manipulate time zones, including, retrieving time zones with abbreviations (“timeZoneWithAbbreviation”), retrieving time zones with names (“timeZoneWithName”), and setting the default time zone (“setDefaultTimeZone”).[19] It was Apple’s creative decision to organize the time zone programs in this manner, select time zone programs that it believed was desirable to programmers and label the time zone programs as they chose.

Likewise, Microsoft provides similar functionality, but with an entirely different structure, naming scheme, and selection. In its Windows Phone development platform, Microsoft stores its time zone programs in the “TimeZoneInfo” class in its “System” namespace (Microsoft’s version of a “package” or “framework”).[20] Within that organizational structure, Microsoft has programs to, among other things, convert time from different time zones (“ConvertTime”) or determine whether a particular date and time in a particular time zone is ambiguous (“IsAmbiguousTime”).[21]

---

[19] *Id.*

[20] TimeZoneInfo Class (System), Windows Phone Dev Center, http://msdn.microsoft.com/en-us/library/windowsphone/develop/system.timezoneinfo(v=vs.105).aspx (last visited Feb. 18, 2013).

[21] *Id.*

As demonstrated by two other major players in the mobile phone operating system industry, the organizational conventions, naming schemes, and selection of programs associated with the concept of using time zones are creative. The ultimate choice for a particular design among an infinite number of alternatives reflects the expression of the Java package designer's judgment as to what design would resonate with programmers and make the Java platform elegant and easy to learn and memorize.

Apple and Microsoft made different creative choices from those found in Java. This difference in creative choices exists amongst Sun/Oracle, Apple, Microsoft, and countless other developers of programming environments--with only one notable exception: Google's undisputed and intentional copying of Java's creative choices at issue in this case. While Sun/Oracle, Apple, and Microsoft invested considerable resources and valuable time making creative decisions for their respective programming libraries, Google did not: it merely copied desirable packages from the Java platform.

The art of package library design stretches beyond the level of individual packages. Because Java programmers need to be familiar with the Java Package library, and because many packages are interrelated, significant attention was paid to the totality of the Java Package library: the selection and arrangement of those packages must be just as appealing and elegant as each individual

package. Making these thousands of creative choices was a massive creative task. Judging from the success of Java (and Google's efforts to copy the naming and organization of Java's packages), the creative choices by Java's architects were brilliant.

III. JAVA LICENSES ITS TECHNOLOGY TO ADVANCE THE WRITE ONCE, RUN ANYWHERE PRINCIPLE

Since its inception, Sun licensed Java to a broad range of technology firms. Java is licensed in three principal ways: under the Gnu Public License ("GPL") v2 License, a commercial license, and the Specification License. These licenses seek to advance the "Write Once, Run Anywhere" principle by ensuring code compatibility and preventing fragmentation. Like the Java Packages, Java's licensing program is essential to maintaining the integrity of the Java framework and safeguarding Java's "Write Once, Run Anywhere" promise.

A. GPLv2 License

The GPLv2 License is an "open source" license. Although commonly perceived to be "free," the GPLv2 License is an enforceable agreement where the licensee's consideration is not monetary, but rather a non-monetary contribution. Under the GPLv2 License, a licensee is required to contribute back to the broad Java community all improvements or changes to the licensed Java technology, guaranteeing that those improvements benefit all users and can be used by all other

licensees. Moreover, this license ensures that changes or improvements cannot be misappropriated for the benefit of a single developer or firm. Thus, as a practical matter, the GPL License ensures compatibility because no GPL licensee has a different or more advanced code-base than any other. The GPLv2 License thus promotes the aim of cross-platform compatibility.

B. Commercial License

Oracle (and previously Sun) also licenses Java under a commercial license, for a fee. For a variety of reasons, many commercial enterprises may choose not to use GPL code in their products and thus opt for a commercial license. The list of commercial Java licensees is long and includes some of the world's foremost technology companies, including Sony, Cisco, RIM, Nokia, Amazon, eBay, Panasonic, LG, Samsung, VISA, and GE. Oracle and a licensee negotiate the terms of each commercial license. In return for a license fee, a licensee is entitled to use all the implementation code and add and modify the code as it pleases. The licensee is also entitled to use the Java trademark and brand. Unlike the GPLv2 License, none of the improvements or alternations have to be passed on to the public at large. But, pursuant to these commercial licenses, a licensee's Java implementation must meet Java's compatibility standards to ensure continuing promotion with the "Write Once, Run Anywhere" principle.

C. Specification License

Oracle (as did Sun) also licenses Java under a Specification License. The Specification License permits licensees to create independent implementation of the Java Specification, including all of the Java Packages that are at issue in this case. However, unlike the Commercial License, a Specification licensee is not entitled to use Sun/Oracle's reference code implementations or the Java trademark and brand. The licensee must write its own code implementing the Java Specification. That new implementing code must pass a compatibility test and must produce the same "specified" end result so that a program written using the standard Java platform would "compute the same result on all machines and in all implementations."[22]

The Specification License is central to preserving Java's "Write Once, Run Anywhere" philosophy. The relevant terms of the license specify that the license is granted on the conditions that the Licensee:

> (i) fully implements the Spec(s) including all its required interfaces and functionality; (ii) does not ***modify, subset, superset*** or otherwise extend the Licensor Name Space, or include any public or protected packages, classes, Java interfaces, fields or methods within the Licensor Name Space other than those required/authorized by the Specification or Specifications being implemented; and

[22] James Gosling, Bill Joy and Guy Steele, *The Java Language Specification, Preface to the First Edition*, July 1996 *available at* http://titanium.cs.berkeley.edu/doc/java-langspec-1.0/j.preface.html (last visited on Feb. 13, 2013).

> (iii) passes the [Technology Compatibility Kit] … for such Specification."[23]

To superset the specification is to exceed the capabilities of the standard Java specification such that the added superset components would not work with the standard Java specification. To subset the specification is to omit certain capabilities of the original specification so that any program designed under the full standard Java specification will not operate with new the subset specification. By restricting Specification Licenses in these ways, Sun (and later Oracle) ensured that all Java code developed pursuant to a Specification License would still be interoperable by being able to run on systems with a standard Java Virtual Machine installation (and thus is faithful to the Write Once, Run Anywhere principle).

IV. GOOGLE'S SUBVERSION OF WRITE ONCE, RUN ANYWHERE

Unlike Apple or Microsoft, who are developing programming platforms for specific mobile devices, Google's Android platform takes a page from Java's "Write Once, Run Anywhere" model and is promoted as the programming platform for all mobile devices. However, Google is unfairly leveraging the Java Packages for its own commercial gain. Google had an opportunity to utilize a Java Specification License for Android. It, nonetheless, directly rejected the terms of

---

[23] Java 2, Standard Edition Specification, June 25, 2003, http://docs.oracle.com /javase/1.4.2/docs/relnotes/license.html (last visited on Feb. 13, 2013) (emphasis added).

that license by copying Java's package names and organization, but then both subsetting and supersetting the specifications in violation of the Specification License.

Despite countless attempts by many levels of Sun employees, including senior executives of Sun, Google declined to license the Java Packages from Sun or Oracle under any of the three available licenses described above. Google did not desire compatibility and it did not want to contribute back any changes to the Java community. Instead, Google merely copied Java's packages that were most useful for mobile platforms, and declined to ensure compatibility with Java. By copying these package elements, names and organization verbatim, Google was able to provide a programming platform that Java programmers had already learned. Indeed, the copying of Java was specifically marketed by Google: Google's own Android developer website states: "*Application Fundamentals - Android applications are written in the Java programming language.*"[24]

Google's own platform, Android, however, was not compatible with Java. Google added other packages that developers used to create programs for Google's Android system. Google also omitted many Java packages. By adding some

---

[24] *See* Application Fundamentals, App Components - Android Developers, http://developer.android.com/guide/topics/fundamentals.html (last visited Feb. 18, 2013).

packages and omitting others, Google ensured that programs written for Android could note be run on any other computer platforms or devices and ensured that programs written for standard Java devices could not be run on Android.

By copying the creative elements of the Java platform familiar to Java developers, but at the same time ensuring that Java code written for Android was transformed into Android-specific code, Google's actions had two consequences: quick access to Java developers while ensuring that Java cross-platform compatibility was not maintained. Google's copying of the creative structure of Java was certainly successful in attracting developers quickly, but undermined the very principle that Java sought to promote: cross-platform compatibility. Because of Google's (largely uncommunicated) changes, additions, and subtractions from its copying of the Java platform, developers who chose to write for Android in using Java platform conventions found that their resulting applications would only work on Google's Android devices. In essence, Google has used the creative aspects of Java to undermine its core mission: "Write Once, Run Anywhere."

When Sun developed the Java platform, it unified disparate computers systems and devices globally under its "Write Once, Run Anywhere" programming platform. Java's innovation made possible the proliferation of a number of computing platforms and devices and helped the Internet grow from a scientific curiosity to an economic and educational engine. Sun and later Oracle invested a

great deal of time, manpower and resources into the Java platform at great monetary risk. By using the Java Packages in violation of the Specification License, Google is reopening the chaos of system fragmentation that the Java platform was meant to stem. The threat that a competitor like Google could simply take the naming conventions and organization of the Java Packages would have deterred Sun from maintaining its decades-long mission to revolutionize computer software development.

Figure B - Summary of Methods in java.text Package

Interface	Method
AttributedCharacterIterator	getRunStart
	getRunLimit
	getAttributes
	getAllAttributeKeys
CharacterIterator	First
	Last
	Current
	Next
	previous
	setIndex
	getBeginIndex
	getEndIndex
	getIndex
	Clone

Class	Method
Annotation	getValue
	toString
AttributedCharacterIterator.Attribute	Equals
	getName
	hashCode
	readResolve
	toString
AttributedString	addAttribute
	getIterator
Bidi	baseIsLeftToRight
	createLineBidi
	getBaseLevel
	getLevelAt
	getRunCount
	getRunStart
	getRunLimit
	requiresBidi
	reorderVisually
	toString
Class BreakIterator	clone
	First

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
	Last
	Next
	previous
	following
	preceding
	isBoundary
	current
	getText
	setText
	getWordInstance
	getLineInstance
	getCharacterInstance
	getSentenceInstance
	getAvailableLocales
	getLong
	getInt
	getShort
ChoiceFormat	applyPattern
	toPattern
	setChoices
	getLimits
	getFormats
	format
	parse
	nextDouble
	previousDouble
	clone
	hashCode
	equals
	nextDouble
CollationKey	compareTo
	equals
	hashCode
	getSourceString
	toByteArray
Collator	getInstance
	compare
	getCollationKey
	equals
	getStrength
	getDecomposition

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
	setDecomposition
	getAvailableLocales
	clone
	equals
	hashCode
DateFormat	format
	public
	parseObject
	getTimeInstance
	getDateInstance
	getDateTimeInstance
	getInstance
	getAvailableLocales
	setCalendar
	setNumberFormat
	setTimeZone
	setLenient
	isLenient
	hashCode
	equals
	clone
DateFormat.Field	ofCalendarField
	getCalendarField
	readResolve
DateFormatSymbols	getEras
	setEras
	getMonths
	setMonths
	getShortMonths
	setShortMonths
	getWeekdays
	setWeekdays
	getShortWeekdays
	setShortWeekdays
	getAmPmStrings
	setAmPmStrings
	getZoneStrings
	setZoneStrings
	getLocalPatternChars
	setLocalPatternChars
	clone

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
	hashCode
	equals
DecimalFormatSymbols	getZeroDigit
	setZeroDigit
	getGroupingSeparator
	setGroupingSeparator
	getDecimalSeparator
	getPerMill
	setPerMill
	getPercent
	setPercent
	getDigit
	setDigit
	getPatternSeparator
	setPatternSeparator
	getInfinity
	setInfinity
	getNaN
	setNaN
	getMinusSign
	setMinusSign
	getCurrencySymbol
	setCurrencySymbol
	getInternationalCurrencySymbol
	setInternationalCurrencySymbol
	getCurrency
	setCurrency
	getMonetaryDecimalSeparator
	setMonetaryDecimalSeparator
	clone
	equals
	hashCode
FieldPosition	getFieldAttribute
	getField
	getBeginIndex
	getEndIndex
	setBeginIndex
	setEndIndex
	equals
	hashCode
	toString

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
Format	format
	formatToCharacterIterator
	parseObject
	clone
Format.Field	
MessageFormat	setLocale
	getLocale
	applyPattern
	toPattern
	setFormatsByArgumentIndex
	setFormats
	format
	formatToCharacterIterator
	parse
	parseObject
	clone
	equals
	hashCode
MessageFormat.Field	readResolve
NumberFormat	format
	parseObject
	parse
	isParseIntegerOnly
	getInstance
	getNumberInstance
	getIntegerInstance
	getCurrencyInstance
	getPercentInstance
	getAvailableLocales
	hashCode
	equals
	clone
	isGroupingUsed
	setGroupingUsed
	getMaximumIntegerDigits
	setMaximumIntegerDigits
	getMinimumIntegerDigits
	setMinimumIntegerDigits
	getMaximumFractionDigits
	setMaximumFractionDigits

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
	getMinimumFractionDigits
	setMinimumFractionDigits
	getCurrency
	setCurrency
NumberFormat.Field	readResolve
ParsePosition	getIndex
	setIndex
	setErrorIndex
	getErrorIndex
	equals
	hashCode
	toString
RuleBasedCollator	getRules
	getCollationElementIterator
	compare
	getCollationKey
	clone
	equals
	hashCode
SimpleDateFormat	set2DigitYearStart
	get2DigitYearStart
	format
	formatToCharacterIterator
	parse
	toPattern
	toLocalizedPattern
	applyPattern
	applyLocalizedPattern
	getDateFormatSymbols
	setDateFormatSymbols
	clone
	hashCode
	equals
StringCharacterIterator	setText
	first
	last
	setIndex
	current
	next

Figure B (Cont.) - Summary of Methods in java.text Package

Class	Method
	previous
	getBeginIndex
	getEndIndex
	getIndex
	equals
	hashCode
	clone

Dated: February 22, 2013

Respectfully submitted,

By: /s/ Steven T. Cottreau
STEVEN T. COTTREAU
CLIFFORD CHANCE US LLP
2001 K STREET, NW
WASHINGTON, DC 20006
(202) 912-5000
Steve.Cottreau@CliffordChance.com
Counsel of Record

Counsel for Amici Curiae Scott Mcnealy and Brian Sutphin

CERTIFICATE OF COMPLIANCE WITH FED. R. APP. P. 29 AND 32

1. This Amici Curiae Brief complies with the type-volume limitations of Federal Rules of Appellate Procedure 29(d) and 32(a)(7)(B) because it contains 6,096 words, excluding the parts of the Brief exempted by Fed. R. App. P. 32(a)(7)(B)(iii).

2. This Amici Curiae Brief complies with the typeface requirements of Federal Rules of Appellate Procedure 32(a)(5) and the type style requirements of Rule 32(a)(6) because this has been prepared in a proportionally spaced typeface using Microsoft Word 2007 14-point Times New Roman font.

Dated: February 22, 2013

/s/ Steven T. Cottreau
Steven T. Cottreau (Counsel of Record)
Counsel for Amici Curiae

CERTIFICATE OF SERVICE

I hereby certify that on February 22, 2013, I electronically filed a copy of this Amici Curiae Brief of Scott McNealy and Brian Sutphin in Support of Plaintiff-Appellant with the Clerk of the Court for the United States Court of Appeals for the Federal Circuit using the Appellate CM/ECF system, which will automatically send email notification of such filing to the following counsel of record:

E. Joshua Rosenkranz, -
Direct: 212-506-5380
Email: jrosenkranz@orrick.com
Fax: 212-506-5151
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, NY 10019

Dale M. Cendali, -
Direct: 212-446-4800
Email: dale.cendali@kirkland.com
Kirkland & Ellis LLP
601 Lexington Avenue
Citigroup Center
New York, NY 10022

Dorian Estelle Daley, Esq., General Counsel
Direct: 650-506-5200
Email: dorian.daley@oracle.com
Fax: 650-506-7114
Oracle America, Inc.
Oracle Legal Department
500 Oracle Parkway
Redwood Shores, CA 94065

Kelly M. Daley, Esq.
Direct: 212-506-5000
Email: kdaley@orrick.com
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, NY 10019

Mark S. Davies, Esq., -
Direct: 202-339-8631
Email: mark.davies@orrick.com
Fax: 202-339-8500
Orrick, Herrington & Sutcliffe LLP
Columbia Center
1152 15th Street, N.W.
Washington, DC 20005

Susan M. Davies
Direct: 202-879-5000
Email: susan.davies@kirkland.com
Kirkland & Ellis LLP
655 15th Street, N.W.
Washington, DC 20005

Sean Fernandes
Direct: 650-859-7014
Email: sean.fernandes@kirkland.com
Kirkland & Ellis LLP
3330 Hillview Ave.
Palo Alto, CA 94304

Annette Louise Hurst, Esq.
Direct: 415-773-4585
Email: ahurst@orrick.com
Orrick, Herrington & Sutcliffe LLP
405 Howard Street
San Francisco, CA 94105

Michael Allen Jacobs, Attorney
Direct: 415-268-7455
Email: mjacobs@mofo.com

Fax: 415-268-7522
Morrison & Foerster LLP
Firm: 415-268-7178
425 Market Street
San Francisco, CA 94105

Kenneth Alexander Kuwayti, Esq.
Direct: 650-813-5600
Email: KKuwayti@mofo.com
Morrison & Foerster LLP
Firm: 650-813-5600
755 Page Mill Road
Palo Alto, CA 94304-1018

Elizabeth Cincotta McBride
Direct: 650-614-7377
Email: emcbride@orrick.com
Orrick, Herrington & Sutcliffe LLP
1000 Marsh Road
Menlo Park, CA 94025

Deborah Kay Miller, Associate General Counsel
Direct: 650-506-0563
Email: deborah.miller@oracle.com
Fax: 650-506-7114
Oracle America, Inc.
Oracle Legal Department
Room 5 op 762
500 Oracle Parkway
Redwood Shores, CA 94065

Gabriel Morgan Ramsey, Attorney
Direct: 650-614-7400
Email: gramsey@orrick.com
Fax: 650-614-7401
Orrick, Herrington & Sutcliffe LLP
1000 Marsh Road
Menlo Park, CA 94025

Matthew Sarboraria, Esq., Associate General Counsel
Direct: 650-506-1372
Email: matthew.sarboraria@oracle.com
Oracle America, Inc.
Oracle Legal Department
m/s 5OP726
500 Oracle Parkway
Redwood Shores, CA 94065

Andrew D. Silverman, Attorney
Direct: 212-506-3727
Email: asilverman@orrick.com
Fax: 212-506-5151
Orrick, Herrington & Sutcliffe LLP
51 West 52nd Street
New York, NY 10019

Joshua L. Simmons, Attorney
Direct: 212-446-4989
Email: joshua.simmons@kirkland.com
Fax: 212-446-4900
Kirkland & Ellis LLP
601 Lexington Avenue
Citigroup Center
New York, NY 10022

Andrew C. Temkin
Direct: 650-506-9432
Email: andrew.temkin@oracle.com
Fax: 650-506-7114
Oracle America, Inc.
Oracle Legal Department
500 Oracle Parkway
Redwood Shores, CA 94065

Diana M. Torres
Direct: 213-680-8400
Email: diana.torres@kirkland.com
Fax: 213-680-8500
Kirkland & Ellis LLP

333 S. Hope Street
Los Angeles, CA 90071

Christa M. Anderson
Direct: 415-391-5400
Email: canderson@kvn.com
Fax: 415-397-7188
Keker & Van Nest LLP
633 Battery Street
San Francisco, CA 94111

Steven A. Hirsch, Attorney
Direct: 415-391-5400
Email: shirsch@kvn.com
Fax: 415-397-7188
Keker & Van Nest LLP
633 Battery Street
San Francisco, CA 94111

Daniel E. Jackson, Esq., Attorney
Direct: 415-391-5400
Email: djackson@kvn.com
Fax: 415-397-7188
Keker & Van Nest, LLP
710 Sansome Street
San Francisco, CA 94111

Michael Soonuk Kwun, Esq., Senior Counsel
Direct: 415-391-5400
Email: mkwun@kvn.com
Fax: 415-397-7188
Keker & Van Nest, LLP
710 Sansome Street
San Francisco, CA 94111

Robert A. Van Nest, Esq.
Direct: 415-391-5400
Email: rvannest@kvn.com
Fax: 415-397-7188
Keker & Van Nest, LLP

710 Sansome Street
San Francisco, CA 94111

Dated: February 22, 2013

/s/ Steven T. Cottreau
Steven T. Cottreau (Counsel of Record)
Counsel for Amici Curiae