No. 18-956

IN THE

Supreme Court of the United States

GOOGLE LLC,

Petitioner,

v.

ORACLE AMERICA, INC.,

Respondent.

On Writ of Certiorari to the United States Court of Appeals for the Federal Circuit

BRIEF OF FORMER SUN EXECUTIVE SCOTT MCNEALY AS *AMICUS CURIAE* IN SUPPORT OF RESPONDENT

CELESTE L.M. KOELEVELD
Counsel of Record
JOHN P. ALEXANDER
YOUNG JUN CHOI
CLIFFORD CHANCE US LLP
31 W. 52nd St.
New York, New York 10019
(212) 878-8000
Celeste.Koeleveld@CliffordChance.com

Counsel for Amicus Curiae

February 19, 2020

WILSON-EPES PRINTING CO., INC. – (202) 789-0096 – WASHINGTON, D. C. 20002

TABLE OF CONTENTS

TABLE OF AUTHORITIES

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

TABLE OF AUTHORITIES—Continued

INTEREST OF *AMICUS CURIAE*[1]

Amicus Curiae Scott McNealy is a former executive of Sun Microsystems, Inc., who was integrally involved in the development and widespread adoption of the Java platform. Sun, 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, an object-oriented, platform-independent, multithreaded programming environment that revolutionized computer programming and quickly became integral to the modern internet.

Mr. 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 CEO from December 1984 to April 2006. In these roles, he worked to make Sun an innovative leader in the information-technology industry.

In 2010, Oracle Corp. acquired Sun, and Mr. McNealy moved on to other projects. While Mr. McNealy does not have any current involvement with Oracle or Java, he shares a vital interest in protecting Java's creative legacy.

---

[1] Pursuant to Supreme Court Rule 37.3(a), *Amicus Curiae* states that counsel for all parties consented to the filing of this brief. Pursuant to Supreme Court Rule 37.6, *Amicus Curiae* states that no counsel for any party authored this brief in whole or in part, and that no person or entity aside from counsel for *Amicus Curiae* made any monetary contribution intended to fund the preparation or submission of this brief.

Because of his close involvement with the creation and widespread adoption of Java, Mr. McNealy submits this brief to explain from his first-hand perspective how Java was the result of exceptional creative efforts by a team of developers and how Google's unauthorized copying of Java's copyrighted code was not fair use.

SUMMARY OF ARGUMENT

This case is about Google's unlicensed copying of approximately 11,000 lines of Java SE code written and copyrighted by Sun/Oracle. The Federal Circuit held that the code at issue was copyrightable, *Oracle Am. Inc. v. Google Inc.*, 750 F.3d 1339 (Fed. Cir. 2014) ("*Oracle I*"), and that Google's infringement was not "fair use," *Oracle Am. Inc. v. Google LLC*, 886 F.3d 1179 (Fed. Cir. 2018) ("*Oracle II*"). These decisions should be affirmed.

The Java platform has evolved and thrived since Sun developed it nearly 25 years ago. Central to this success was Sun's inclusion of "packages" of pre-written programs that allow programmers to code quickly and efficiently through a series of elegant shortcuts. These programs operate by means of declaring code (which describes the pre-written program and is used to invoke it) and implementing code (which performs the requested function). Designing these packages and the related declaring code involved a massive creative effort by professionals at Sun, who had to decide, among other things, what packages to create and how to name and organize them in a way that resonated with programmers. Although the declaring code used to invoke the programs 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 an architect can see lines in a blueprint as an iconic silhouette in a city skyline. The declaring code in Java comes intuitively to programmers, and is a huge part of what has made Java such a success.

In this case, Google copied verbatim the declaring code and related structure of 37 Java packages for its Android smartphone operating system. Google took the declaring code and structure of the 37 packages specifically because that code was popular among developers and was what they would expect to use.

Google argues in part that this code is not copyrightable because it is merely functional, like labels in a file cabinet. The idea is that no one should be able to copyright what is (supposedly) the mere organization of generic folders and drawers. But this is a woeful mischaracterization of the artful design of the Java packages, and is an insult to the hard-working developers at Sun who made Java such a success. Sun had unlimited options in writing the Java packages, and made creative choices so that the code would resonate with programmers and be intuitive to how they think. The declaring code is therefore less like a file cabinet, and more like a detailed table of contents, with chapter and subchapter headings, and topic sentences that forms an integral part of a well-written book. It is this very elegance that made Java popular, and gave Google the incentive to copy it.

Google also argues that the declaring code is not copyrightable because there was only one way of expressing that code in the specific way most familiar to Java developers. But contrary to Google's sugges-

tion, the popularity of a work is no reason for it to lose copyright protection.

Furthermore, Google's arguments regarding "fair use" fail for the reasons articulated by the Federal Circuit and in Oracle's brief. *Amicus* McNealy wishes to address herein only certain points relevant to his unique position at Sun:

First, contrary to Google's suggestion, Sun always required a license for the commercial use of Java SE.

Second, Google's use of the Java SE code in Android was not a new context for Java, which was being used in smartphones before Android.

Third, Google's use of Java SE was particularly unfair because it did so in a way that rendered Android incompatible with Java. Thus, Google's use not only made Sun/Oracle suffer financially, but also destroyed Java's promise of cross-compatibility.

The Federal Circuit should be affirmed.

ARGUMENT

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 a few common rules and vocabularies in writing programs for different computer systems and devices. Each device, however, had unique requirements, and so each program had to be written in a manner specific to a particular device. Although a programmer could write programs for multiple systems or devices in the same language, the program itself would have to be rewritten (or "ported") for each type of computing device. For example, an application written in the programming language "C" and designed for a Microsoft Windows PC would not work on any non-Windows device.

Having to port programs for multiple systems slowed the process of making software available for new platforms and devices. It was also often prohibitively expensive. Developers therefore often chose to write software for systems with the largest number of users. Early efforts to develop a cross-system platform failed.[2]

Innovators at Sun realized they needed to start "really [bearing] down and . . . help customers solve the problems they were having in migrating away from mainframes."[3] A team of Sun computer engi-

---

[2] James Daly, *Apple, Symantec Rethink Role Bedrock Will Play*, Computerworld, Dec. 20, 1993, at 69.

[3] Lee Gomes, *Made in the Shade; 'Java' Stirs Up Renewed Interest in Sun Micro*, Dall. Morning News, Dec. 18, 1995.

neers, led by James Gosling, began developing a computer platform intended to revolutionize how people would program.

The Java platform was the result: a paradigm-shifting platform that permitted developers to "Write Once, Run Anywhere." Developers could write a program once, using Java, and the program could run on a variety of computing systems and devices. This was therefore an inspirational breakthrough in software development.[4]

The Java platform proved an enormous success. Java has evolved and thrived for nearly 25 years while competing against myriad other development platforms. Java's success not only advanced "Write Once, Run Anywhere," but it was at the forefront of the 1990s internet revolution. Indeed, commentators at the time referred to Java as the "hottest idea in the high-tech world," and some touted a "vision of how a Java-charged Web could change the computing industry."[5] Internet-defining companies, such as Twitter and Netflix, rely on Java for the infrastructure to power their businesses.[6] And as discussed *infra*, Java opened up the world of mobile phones and devices to the possibilities of the internet.

---

[4] David Bank, *The Java Saga*, Wired Magazine (Dec. 1, 1995), https://www.wired.com/1995/12/java-saga/ (last visited Feb. 11, 2020).

[5] Elizabeth Corcoran, *Java Jumps Into the Net*, Wash. Post, Dec. 10, 1995.

[6] *Java Powers Our Digital World*, Java, https://go.java/?intcmp=gojava-banner-java-com (last visited Feb. 7, 2020).

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 called the Java Packages,[7] which developers can use to speed the creation of new applications; (iii) a Java compiler, which translates developer-written code into byte-code; (iv) a Java Virtual Machine ("JVM"), which translates Java byte-code into instructions comprehensible to the underlying computing device; and (v) a Java Development Kit ("JDK"), a collection of programming tools released by Oracle. Only Google's copying of the Java Packages is at issue in this case.

Java Language. The Java language constitutes the "bare bones" of the Java framework, providing the syntax, grammar, and vocabulary that permits a developer to write programs in Java. This language is not identical to the Java platform or the various programs written in Java. For instance, the Java Packages (described below) are written in Java, but Java can also be used to write different programs achieving the same or similar functionality.

Java Packages. The Java Packages are an extensive set of ready-to-use programs that serve as "building blocks" for Java developers. These pre-built packages allow programmers to accomplish tasks ranging from the simple (like basic math functions) to the complex (like computer security and network access) without having to write their own code to do so. The Java Packages thus provide various shortcuts so

---

[7] The Java Packages are sometimes also called application programming interfaces ("APIs") or the Java Class Library.

that Java developers need not "reinvent the wheel" for many desired programming tasks.

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 used 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 is connected to a locally networked computer, to retrieving the IP address of another computer connected to the internet.

The Java Packages operate with essentially two types of source code: declaring code and implementing code. The declaring code is used to invoke the prewritten program—it functions like a name and description of the program's place in the package hierarchy. The implementing code then tells the computer how to perform the declared function. Pet. App. 216a, 221a–225a.

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 is distributed to end-users to run on different computing platforms.

JVM. Installed on a user's device, the JVM is software that translates the Java byte-code to enable it to run on a computer or device. Once a JVM exists for a platform, all devices using that type of platform can run programs written in Java.

JDK. The JDK is the culmination of all the elements of the Java platform. It provides all the necessary and optional programs and tools to develop and test Java applications, including the Java Packages, the Java Compiler, and the JVM.

II. THE JAVA PACKAGES ARE CREATIVE AND EXPRESSIVE

At issue here is Google's verbatim copying of the declaring code and organization of 37 Java Packages. Google argues that this material is not copyrightable essentially because (i) the code is a purely functional method of operation, *i.e.*, it is not expressive; and (ii) there is only one way of expressing that function in the way most familiar to Java developers. Google Br. 13–15. Google is wrong on both counts.

A. Designing an Elegant Set of Packages Was Central to Java's Success

It is well-settled that the Copyright Act applies to computer programs.[8] *See, e.g., Atari Games Corp. v. Nintendo of Am., Inc.*, 975 F.2d 832, 838 (Fed. Cir. 1992) ("As literary works, copyright protection extends to computer programs[.]"); 1 Melville B. Nimmer and David Nimmer, *Nimmer on Copyright* § 2A.10[B] (2020). Thus, the Java code and organization are protectable as a general matter if they are expressive and have "at least some minimal degree of creativity." *Feist Publ'ns, Inc. v. Rural Tel. Serv. Co.*, 499 U.S. 340, 345 (1991). As described below, this standard is easily satisfied here.

---

[8] The statute defines a "computer program" as "a set of statements or instructions to be used directly or indirectly in a computer to bring about a certain result." 17 U.S.C. § 101.

As noted above, the Java Packages provide a set of pre-existing programs that allow developers to quickly and efficiently create their own applications in Java. But merely offering a programming platform in which developers could build cross-system applications would not have been enough to get developers to use it. Indeed, that functionality could have been made accessible in any number of ways (as discussed *infra*). 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, memorize, and master. A package's declaring code therefore must communicate what a program does, how it relates to other programs, and what is needed for it to work, in a way that is intuitive and resonates with programmers. For this reason, Sun invested an enormous amount of creativity in developing an original hierarchy of packages organized into a distinctive library and unique classes. This code and organization is integral to the program, the way a table of contents is intertwined with a book.

Designing a package entailed numerous creative choices. Java's architects had to decide what packages to create, and within those, how many classes and methods to include. Moreover, for every package, class, and method that made the cut, Java's creators had to name each element and determine how to arrange them. Over time, Java's developers evolved existing packages and added new ones.[9] And the art of

[9] Developing a proposed package can take over a year. The process involves multiple levels of review, including input from experts and an opportunity for public feedback. The final release must pass various tests, most importantly compatibility with the existing Java Packages. *JCP 2.11: Process Document*, Java

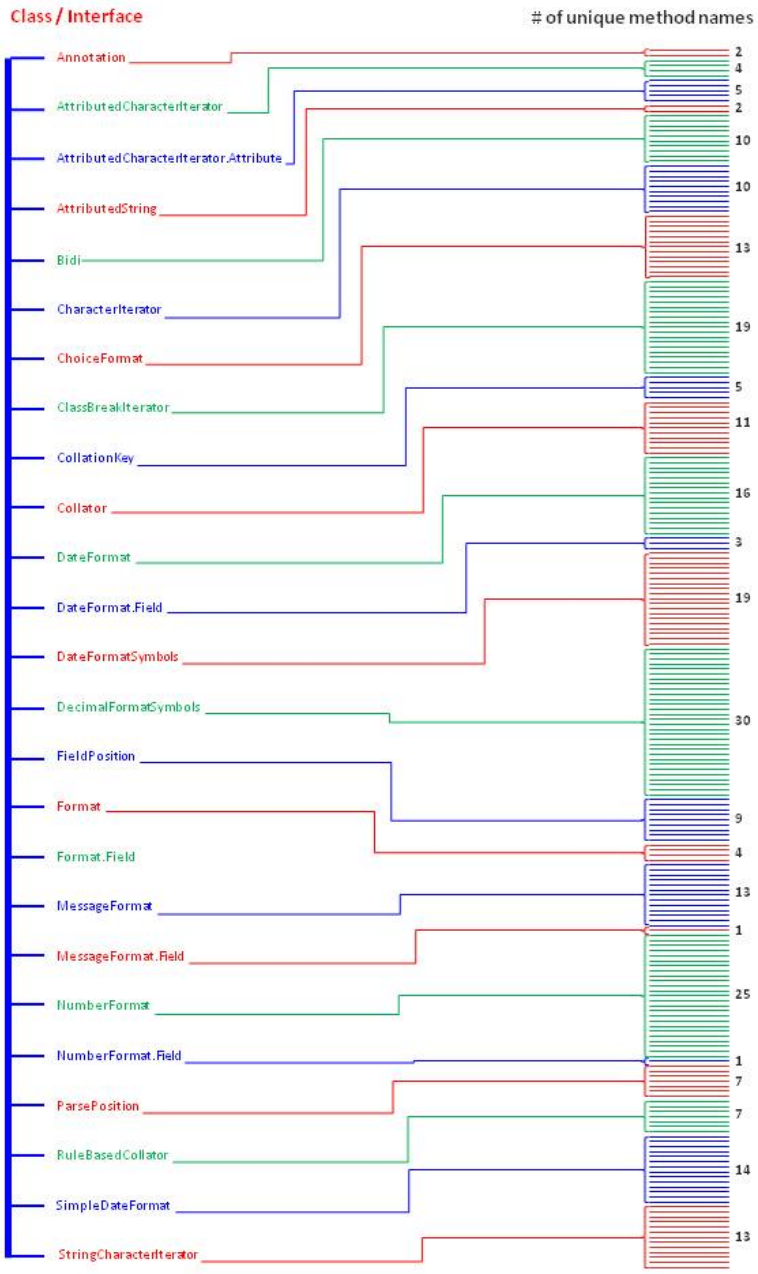
package design stretches beyond the level of individual packages. Because Java programmers benefit from familiarity with the Java Package library, and because many packages are interrelated, significant attention was paid to the totality of the library: the selection and arrangement of those packages must be just as appealing and elegant as each individual package. Making these thousands of choices was a massive creative task. *Oracle I*, C.A. 20,797–98. None was required for the programs to perform their function or dictated by the Java language. Pet. App. 165a–166a.

An examination of the java.text package from Java Standard Edition 5.0 illustrates the myriad design choices the Java package creators faced. The java.text package contains 25 classes, two interfaces, and hundreds of methods to handle text, dates, numbers, and messages in a manner independent of natural human languages (allowing for different localized uses).[10] Figure A sets forth an overview of this Package's elaborate class structure (which does not show the additional detail of method names).

Cmty. Process, https://www.jcp.org/en/procedures/jcp2_11#3.2 (last visited Feb. 11, 2020).

[10] *Package java.text*, Oracle, https://docs.oracle.com/javase/1.5.0/docs/api/java/text/package-summary.html (last visited Feb. 11, 2020).

Figure A- java.text Package



Java's creators initially needed to determine whether to include a java.text package in the first place. Indeed, the localization functions offered by the java.text package could have been grouped in other ways, but Java developers believed that combining them in a single package worked best.

That, however, was just the start of thousands of such choices that went into drafting a 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 unique and easy to grasp names.

Beyond the names, Java's developers had to make appealing organizational choices. For example, organizing the java.text package alphabetically or chronologically were possibilities, but not choices that Java's creators made. The class names and organization could not be so lengthy that they would be inefficient for programmers to use regularly, yet the labels still needed to convey the package's purpose.

Moreover, Java's architects needed to consider how java.text would be interrelated with the broader Java Package library. Java.text is utilized by eight other packages, including packages dedicated to fonts, user interfaces, and images.

All of the above-described choices are creative, not merely functional. Other names and organizational structures could have been used to invoke the pre-written programs at issue (*see infra*). The ultimate choice made by Sun (or Oracle) for a particular design reflected judgment as to what design would resonate

with programmers and make the Java platform easy to learn and memorize.

These choices are important. For example, Google's own Principal Java Engineer Joshua Bloch (who previously worked as Senior Sun Engineer on the Java project) has stressed the importance of creativity in developing a package library, explaining that "APIs can be among a company's greatest assets," and that integral to the process of package design is naming ("Good names drive development").[11] Mr. Bloch has noted that "[n]ames matter" and that, if a programmer creates elegant names, "code will read like prose"; indeed, "API design is an art, not a science. Strive for beauty, and trust your gut."[12]

Likewise, a leading Java educator has explained that "somewhere in the mix of Sun's design objectives, it seems that there was a goal of reducing *complexity for the programmer*."[13] As one programmer 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

[11] Trial Ex. 624 at 3, 14.

[12] Trial Ex. 877 at 2, 3.

[13] Bruce Eckel, *Thinking in Java* 1 (Prentice Hall 4th ed. 2006).

> development cycle coming from different frameworks or libraries. API beauty depends on designer knowledge and design capability (say artistic skill).[14]

Indeed, because of the appeal of Java's naming and organization, a community devoted to the development of new and existing packages was established. The result has been one of the most enduring programming platforms ever conceived.

B. There Are Countless Ways to Achieve the Functionality Provided by the Java Packages

Google argues that the merger doctrine precludes copyright protection for the material Google copied. This argument fails.

The merger doctrine provides that when there is only one or very few ways "to express an idea, [] the expression is said to have 'merged' with the idea" such that it cannot be protected. *Comput. Assocs. Int'l, Inc. v. Altai, Inc.*, 982 F.2d 693, 707–08 (2d Cir. 1992); *Gates Rubber Co. v. Bando Chem. Indus.*, 9 F.3d 823, 838 (10th Cir. 1993).[15] But the notion that there was only one way to write the code at issue is simply wrong.

As an initial matter, the relevant consideration in the merger analysis is whether there were alternative ways of expressing the code *at the time Sun wrote it*. Indeed, copyright in a work "subsists from its creation[.]" 17 U.S.C. § 302(a). Thus, if Sun had many choices when it wrote the Java declaring code, it would

---

[14] Adam Brown, *Beautiful API Design*, DZone (Nov. 26, 2008), https://dzone.com/articles/beautiful-api (last visited Feb. 5, 2020).

[15] The merger doctrine arises out of the rule that copyright does not extend to an idea. *See* 17 U.S.C. § 102(b).

not matter if Google only had one way of writing the declaring code at the time of its copying. But either way, in this case, there were multiple ways of writing the code at issue both when Sun first wrote it and when Google later copied it.

As described *supra*, and as the Federal Circuit found, Sun/Oracle had "'unlimited options as to the selection and arrangement of the [] lines Google copied[.]'" *Oracle I*, 750 F.3d at 1361; *see also id.* at 1368 ("[T]he declaring code could have been written and organized in any number of ways and still have achieved the same function."); *id.* at 1371 ("[T]here is no evidence that when Oracle created the Java API packages at issue it did so to meet compatibility requirements of other pre-existing programs.").

Likewise, it is clear that Google could have designed its own packages by writing original declaring code to call up the new implementing code it wrote. As the District Court acknowledged, "the Android method and class names could have been different from the names of their counterparts in Java and still have worked." Pet. App. 215a; *see also id.* at 266a ("There was nothing in the rules of the Java language that required that Google replicate the same groupings even if Google was free to replicate the same functionality."). The Federal Circuit also found that "nothing prevented Google from writing its own declaring code, along with its own implementing code, to achieve the same result." *Oracle I*, 750 F.3d at 1361; *see also id.* at 1368 ("Google could have structured Android differently and could have chosen different

ways to express and implement the functionality it copied.").[16]

The availability of alternative means of writing the code at issue is illustrated by considering the various approaches taken by other competitors. For example, take the Java Packages' approach to time zones. In Java, a developer setting the time zone in an application would 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 time zone for that application.

But this was not the only way to set a time zone in an application. Apple's iOS platform devotes an entire class to set the time zone—the "NSTimeZone" class. Unlike Java's placement of that function in the java.text package, Apple put it in its "Foundation" framework (Apple's term for a structure conceptually similar to Java's "package"). Apple's NSTimeZone class contains numerous methods to manipulate time zones, including through abbreviations ("timeZoneWithAbbreviation"), names ("timeZoneWithName"), and default settings ("defaultTimeZone").[18] It was

---

[16] Similar programs could have been written in the Java language with different declaring code. *See* Tr. 290:4–291:9, ECF No. 943 (Larry Ellison); Tr. 2212:19–2213:16, ECF No. 1065 (Owen Astrachan).

[17] *Class DateFormat*, Oracle, https://docs.oracle.com/javase/1.5.0/docs/api/java/text/DateFormat.html (last visited Feb. 7, 2020).

[18] *NSTimeZone*, Apple Developer, https://developer.apple.com/documentation/foundation/nstimezone?language=objc (last visited Feb. 7, 2020).

Apple's decision to organize and label the time zone programs in this manner.

Likewise, Microsoft provided similar functionality, but with an entirely different structure and naming scheme. In its former Windows Phone development platform, Microsoft stored its time zone programs in the "TimeZoneInfo" class in its "System" namespace (Microsoft's version of a "package" or iOS "framework"). Within that structure, Microsoft had 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").[19]

As demonstrated by two other software developers, the organizational conventions, naming schemes, and selection of programs associated with the concept of using time zones are open to various modes of expression. But 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 the declaring code for desirable packages from the Java platform.

What all of this really comes down to is that Google wanted to attract Java developers and get to market quickly. As Google itself explains, such developers "would expect to use" the popular (and copyrighted) Java declaring code; otherwise, they would have had to "learn thousands of new calls." Google Br. 7–8. By copying the elements, names, and organization of the Java Packages verbatim, Google was therefore able to

---

[19] *TimeZoneInfo Class*, Microsoft, https://docs.microsoft.com/en-us/dotnet/api/system.timezoneinfo?view=netframework-4.8 (last visited Feb. 5, 2020).

provide a platform that Java programmers already knew.[20]

Google's merger argument therefore is essentially that there is only "one way" of expressing Java's declaring code in the specific way already made popular by Java. That circular argument would undermine any copyright protection for popular works, which cannot be the law. *See, e.g., Prac. Mgt. Info. Corp. v. Am. Med. Ass'n*, 121 F.3d 516, 520 n.8 (9th Cir. 1997) (rejecting theory that a work loses copyright protection just because a government regulation makes it pervasive); *Educ. Testing Servs. v. Katzman*, 793 F.2d 533, 540 (3d Cir. 1986), ("possible domination in [particular] field . . . cannot excuse copying . . . and patently does not affect the validity of [plaintiff's] copyright"), *abrogated on other grounds*, *TD Bank N.A. v. Hill*, 928 F.3d 259 (3d Cir. 2019).

III. GOOGLE'S USE OF JAVA WAS UNAUTHORIZED

Google suggests that Sun/Oracle did not view the declaring code at issue as proprietary, and/or that there was some purported industry standard to that effect. Nothing could be further from the truth.

---

[20] Google's Android developer website specifically marketed, until at least September 2017, that: "*Android applications are written in the Java programming language.*" *Application Fundamentals*, Android Developers, https://web.archive.org/web/20170919103505/https://developer.android.com/guide/components/fundamentals.html (last visited Feb. 7, 2020). *See also* Tr. 629:24–630:2, ECF No. 1907 (Andy Rubin) (Android's use of already-created software was "a huge accelerant to [its] effort").

A. Sun's Licensing Requirements

Since its inception, Sun has licensed Java to a broad range of technology firms. Java is licensed in three principal ways (briefly summarized below), all meant to advance the "Write Once, Run Anywhere" principle:

- The public license is an "open source" license based on the widely used General Public License ("GPLv2").[21] Although commonly perceived to be "free," it is an enforceable agreement where the licensee must contribute back to the Java community all improvements or changes to the licensed technology, guaranteeing that those improvements benefit all users.
- The Specification License permits licensees to use the Java declaring code (including the declaring code for all of the Java Packages at issue in this case) and structure. However, the licensee must write its own implementing code. That new implementation 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] These licensing terms are spelled out explicitly in the Java API Specification, the manual that details the Java

---

[21] Open-source licenses "are used by . . . software developers . . . who wish to create collaborative projects and to dedicate certain works to the public," while "provid[ing] creators of copyrighted materials a means to *protect and control their copyrights*." *Jacobsen v. Katzer*, 535 F.3d 1373, 1378 (Fed. Cir. 2008) (emphasis added).

[22] James Gosling et al., *The Java Language Specification* xxiii (Lisa Friendly ed., Addison-Wesley 1996).

declaring code and structure.[23] Some of the prominent technology firms that have taken Java's Specification License include IBM, SAP, and Red Hat.[24]

- For a commercial license, a licensee pays a fee for use of all the implementation code and may alter the code as it pleases. While none of the alterations have to be passed on to the public, the implementation must meet Java's compatibility standards. Java's commercial licensees have included the world's foremost technology companies, including Sony, Cisco, RIM, Nokia, Amazon, eBay, Panasonic, LG, Samsung, VISA, and GE.

Beginning in 2005, Google engaged in extensive discussions with Sun about the possibility of licensing Java for use in the Android platform. But despite countless attempts by Sun employees (including senior executives), Google declined to license the Java Packages from Sun or Oracle under any of the available licenses described above.[25] Ultimately, Google decided to (in its own words) "[d]o Java anyway and defend [its] decision, perhaps making enemies along the way." Pet. App. 106a (e-mail from Android founder to Google CEO). The negotiation did not break down over money. Instead, Google's principal reason for not agreeing to a license was that—while it wanted to

---

[23] *Java 2 Platform Standard Edition Development Kit 5.0 Specification*, Oracle (Aug. 25, 2004), https://docs.oracle.com/javase/1.5.0/docs/relnotes/license.html (last visited Feb. 7, 2020).

[24] Tr. 293:8–295:6, 299:11–303:17, ECF No. 943 (Ellison); *JCP Members*, Java Cmty. Process, https://jcp.org/en/participation/members (last visited Feb. 11, 2020).

[25] *See* Tr. 886:3–888:5, ECF No. 1909 (Rubin).

provide a platform that Java programmers had already learned—it wanted neither compatibility between Android and Java SE nor to contribute back any changes to the Java community. *See Oracle II*, 886 F.3d at 1187.

Google cites testimony from certain former Sun executives (including former CEO Jonathan Schwartz)[26] who conclusorily assert their belief that a license was not needed for Java's declaring code. Google Br. 38. But this contradicts the clear terms of the Java Specification manual and Specification License (*see supra*), as well as the actual practice at Sun/Oracle. Indeed, as *Amicus* McNealy testified, while Sun "offered lots of [its] technology for free . . . in terms of no revenue charge," it did so pursuant to a license, and "did not license [its] technology with rights equivalent to ownership, even if it was free and open."[27] For instance, Andy Rubin (who later founded Android) felt the need to obtain a Java license when he launched his own earlier smartphone, the Danger Sidekick.[28] And it is telling that Google points to no instances where Sun actually allowed any commercial use of its declaring code without a license.

Google also asserts that Sun supposedly "did not consider Google's reuse of the declarations to be infringement." Google Br. 38. This is simply not true. For example, in 2007, *Amicus* McNealy (then-Chairman of Sun's board) expressly told Mr. Schwartz that

---

[26] Notably, Oracle did not keep Mr. Schwartz on as CEO after acquiring Sun. Tr. 562:23–565:25, ECF No. 1907 (Jonathan Schwartz).

[27] Tr. 2067:10–68:17, 2077:3–78:2, ECF No. 1064 (Scott McNealy).

[28] Tr. 887:23–889:3, ECF No. 1909 (Rubin).

"[t]he Google thing is really a pain. They are immune to copyright law."[29] Google points to a 2007 blog post from Mr. Schwartz in which he stated, among other things, that Android "strapped another set of rockets" to Java. But Google ignores that in internal communications, Mr. Schwartz was at the same time complaining that he "h[ad] no clue what [Google is] up to" and "[m]y sense is they're playing fast and loose with licensing terms."[30]

B. Google's Flawed "Industry Practice" Argument

Google also says that there was a purported industry understanding that declaring code may be reused, citing the subjective beliefs of its witnesses. Google Br. 38. But this ignores the actual practice at Sun/Oracle, noted above.

Various *amici*—including Java competitors Microsoft and IBM—repeat the argument that industry practice supposedly is to allow re-use of declaring code. But these self-serving arguments come from companies with a long history of trying to thwart Java and/or compete with Sun.

For instance, in the late 1990s, Microsoft attempted to use Java while altering the parts that made it interoperable (*i.e.*, the sort of incompatible "re-implementation" Google did here). Sun sued based on Microsoft's violation of its license agreement. *See, e.g., Sun Microsystems, Inc. v. Microsoft Corp.*, 188 F.3d 1115 (9th Cir. 1999). As the district court noted:

---

[29] Tr. 608:2–609:12, ECF No. 1907 (Schwartz).

[30] *Id*. at 586:1–24 (Schwartz).

> Microsoft's unauthorized distribution of incompatible implementations of Sun's Java Technology threatens to undermine Sun's goal of cross-platform and cross-implementation compatibility. The threatened fragmentation of the Java programming environment harms Sun's relationship with other licensees . . . .

Sun Microsystems, Inc. v. Microsoft Corp., 87 F. Supp. 2d 992, 997 (N.D. Cal. 2000). The case settled, but Sun's disputes with Microsoft continued. *See In re Microsoft Corp. Antitrust Litig.*, 333 F.3d 517, 536 (4th Cir. 2003) (affirming preliminary injunction based on settlement violation). Notably, in the massive *U.S. v. Microsoft* litigation, the D.C. Circuit noted that Microsoft's "ultimate objective was to thwart Java's threat to Microsoft's monopoly," citing a Microsoft document "stat[ing] as a strategic goal: 'Kill cross-platform Java by grow[ing] the polluted Java market.'" *U.S. v. Microsoft Corp.*, 253 F.3d 34, 76 (D.C. Cir. 2001).

IBM also has had various disputes with Sun/Oracle over the years. For instance, in 2000, IBM withheld its endorsement of Sun's Java specification, which analysts said was an attempt to "wrestle some control of Java."[31] And in the late-2000s, IBM backed the Apache Software Foundation in its objections to certain of Java's licensing requirements. This came at around the same time that IBM attempted (unsuccessfully) to acquire Sun.[32] IBM backed down in the

---

[31] Lee Copeland, *Big Blue, Sun in Java Dispute*, JavaWorld (June 23, 2000), https://www.javaworld.com/article/2076110/big-blue--sun-in-java-dispute.html (last visited Feb. 7, 2020).

[32] Jon Swartz & Leslie Cauley, *Oracle to Buy Sun for $7.4B after IBM Drops Bid*, ABC News (Apr. 21, 2009), https://abcnews.

licensing dispute when Oracle acquired Sun and made clear that it would continue Sun's policy. At the time, IBM noted: "We think this is the pragmatic choice. It became clear to us that first Sun and then Oracle were never planning to make the important test and certification tests for Java, the Java SE TCK, available to Apache."[33]

These past disputes underscore Sun's history of zealously protecting its intellectual property rights, and show the self-interest of some of Google's *amici*.

IV. GOOGLE'S USE OF JAVA WAS NOT TRANSFORMATIVE

Google argues that its use of the Java code should be allowed as "fair use" because, among other reasons, Android was supposedly transformative in that it used Java in the smartphone context. This argument fails: Java was designed as a cross-platform system, and Sun had pioneered smartphone uses of Java before Google created Android.

A work is transformative if it does not "merely supersede the objects of the original creation . . . [but] instead adds something new, with a further purpose or different character, altering the first with new expression, meaning, or message." *Campbell v. Acuff-Rose Music, Inc.*, 510 U.S. 569, 579 (1994).

But Google copied the declarations at issue verbatim and without alteration, and used them for the same

---

go.com/Technology/story?id=7395780&page=1 (last visited Feb. 11, 2020).

[33] Ryan Paul, *Java Wars: IBM Joins OpenJDK as Oracle Shuns Apache Harmony*, Arts Technica (Oct. 13, 2010), https://arstechnica.com/information-technology/2010/10/ibm-joins-openjdk-as-oracle-shuns-apache-harmony/ (last visited Feb. 7, 2020).

purpose in Android. In particular, Android developers could use Java's declaring code to invoke the same functionality that this code invoked in the Java Packages.[34] This was by design (as discussed *supra*). The district court recognized "of course, the copied [Java Packages] serve the same function in both works," Java and Android, "for by definition, declaring code in the Java programming language serves [a] specific definitional purpose." Pet. App. 110a. Therefore, Google did nothing more than "supersede the objects of the original creation" without adding any new elements.

Google argues that Android's use of Java was transformative (indeed, "revolutionary") supposedly because Java "was built to run on servers and desktop computers," and was not "suitable for the serious constraints of a smartphone platform." Google Br. 43. This simply ignores Java's long history of device and platform independence.

Java was always intended to work across different kinds of hardware, as described in more detail below. As one Java founder put it, "it was patently obvious" as early as 1993 that Java "fit perfectly with the way applications were written, delivered, and used on the Internet."[35] Indeed, Java was meant to unlock the potential of the internet and mobile computing, giving developers the opportunity to learn one language that could be used across many devices. The internet had

---

[34] The fact that Google wrote its own implementing code to carry out the declared function does not change that the declaration itself is used for the same purpose.

[35] Jon Byous, *Java Technology: An Early History*, Sun Developer Network (Apr. 2003), https://web.archive.org/web/20090311011509/http://java.sun.com/features/1998/05/birthday.html (last visited Feb. 7, 2020).

an early major challenge: many different devices sought to access a given website. Websites are, after all, nothing more than computer programs existing on a remote computer known as a server. Java enabled these computer programs (*i.e.*, websites) to run on any device running a JVM, regardless of the type of device or operating system.[36] The Java platform met this goal by creating a rich ecosystem, closely guarded and nurtured by Sun and, later, Oracle, designed to ensure cross-platform compatibility—Java's core "Write Once, Run Anywhere" value.

A brief look at Java's history with mobile devices is illustrative. For example, as early as 1991, Sun established the "Green Team," a group of 13 people initiated by Patrick Naughton, Mike Sheridan, and James Gosling. The team was tasked with anticipating and planning for the "next wave" in computing.[37] It quickly developed the Star7, a PDA mobile device with an animated touch screen that ran an early version of what would become the Java platform.[38] It could control multiple entertainment devices, such as televisions, VCRs, and stereos, running on a "processor-independent" language, the precursor of Java's JVM environment.[39]

---

[36] Edward Yourdon, *Java and the New Internet Programming Paradigm*, JavaWorld (Aug. 1, 1996), http://www.javaworld.com/article/2077231/java-and-the-new-internet-programming-paradigm.html (last visited Feb. 7, 2020).

[37] Byous, *supra* note 35.

[38] Klint Finley, *Tech Time Warp of the Week: Get a Glimpse of the Lost Touchscreen Tablet of 1992*, Wired (Oct. 17, 2014), https://www.wired.com/2014/10/star7/ (last visited Feb. 7, 2020).

[39] Byous, *supra* note 35.

As Java developed, developers could exploit the possibilities of the internet across a wide variety of connected devices. The Java platform transformed what had been a medium for sending text and images to a vibrant, multimedia environment with nearly infinite possibilities. In 1997, the Java platform was already expanding beyond the internet and traditional desktop PCs to items such as smartcards.[40]

At 1998's JavaOne Developer Conference, attendees received a JavaRing, a wearable device with an embedded microprocessor, which used the Java platform to bring attendees their preferred coffee after being scanned by readers located throughout the conference. The demonstration showed Java's utility in a far reaching array of applications and device form factors, many of which are now routine parts of smartphones, such as exchanging contact information, using banking services, and starting cars.[41]

This focus on mobile devices continued in 1999 with the introduction of the Java 2 Platform, Micro Edition ("J2ME"), a derivative of Java's Standard Edition ("J2SE") designed specifically for mobile devices.[42] Years before Google considered creating Android, Sun had developed an entire infrastructure for the use of the Java platform in mobile computing devices,

---

[40] Rinaldo Di Giorgio, *Smart Cards: A Primer*, JavaWorld (Dec. 1, 1997), https://www.javaworld.com/article/2077101/smart-cards--a-primer.html (last visited Feb. 7, 2020).

[41] Margaret Rouse, *Java Ring*, WhatIs.com (Sept. 2005), https://whatis.techtarget.com/definition/Java-Ring (last visited Feb. 11, 2020).

[42] Chuck McManis, *The Palm V Meets J2ME*, JavaWorld (Aug. 20, 1999), https://www.javaworld.com/article/2076478/the-palm-v-meets-j2me.html (last visited Feb. 7, 2020).

including the Java Phone API.[43] By 2000, Java was ubiquitous in the growing personal communications market, appearing in two-way pagers, mobile phones, and palm computers/PDAs.[44] For instance, as early as 2000, Research in Motion—the company that created the Blackberry smartphone—was using the Java platform to create a programmable, web-enabled pager.[45]

A demonstration at the JavaOne conference in 2002 showcased a mobile phone-controlled robot in a sumo wrestling match with a desktop PC-controlled robot.[46] The message was clear: Java, designed as a cross-platform system, enabled mobile devices to do anything a computer could do. As mobile devices, including smartphones and tablets, grew more sophisticated and had greater processing power, developers realized they could use the Java platform to bring programs that were previously imprisoned in large desktop machines to sleek, portable handheld devices.[47]

As the smartphone industry grew in the early and mid-2000s, Java was used in a variety of pre-Android devices, including the Danger Sidekick (one of the

---

[43] Bill Day, *Program Java Devices – An Overview*, JavaWorld (July 24, 1999), https://www.javaworld.com/article/2076441/java-se-program-java-devices-an-overview.html (last visited Feb. 10, 2020).

[44] *Id*.

[45] Mark Beaulieu, *Wireless Internet Applications and Architecture: Building Professional Wireless Applications Worldwide* 289 (Addison-Wesley 2002).

[46] Tim Rohaly, *Real-Time Java Takes the Stage*, JavaWorld (Mar. 26, 2002), https://www.javaworld.com/article/2073884/real-time-java-takes-the-stage.html (last visited Feb. 9, 2020).

[47] Tr. 1670:9–13, ECF No. 1932 (Alan Brenner).

first, if not the first, smartphones), Blackberry, HTC, Nokia, and SavaJe.[48] *Amicus* McNealy personally negotiated a Java ME license with Motorola, which was, at the time, perhaps the biggest player in the mobile phone industry.[49]

Andy Rubin, the creator of the Danger Sidekick, testified that his company "created our own implementation of the Java 2 SE APIs" and to do so obtained a Java license from Sun. Mr. Rubin further testified that the "rest of the industry . . . was using Java in some of the phones."[50] And after licensing Java for use on smartphones, Mr. Rubin went on to found Android, which was then acquired by Google.[51] Mr. Rubin admitted that Android considered Java a direct competitor that was "targeting the same industry with similar products."[52]

Thus, contrary to mobile phones being a new context, the Java platform already existed in the "operating environment of mobile smartphone devices" before the advent of Android. Pet. App. 111a. Due to its elegance, portability, and functionality, Java dominated the mobile market, including both feature phones[53] (Java ME) and emerging smart-

---

[48] *Id*. at 1622:13–21, 1623:10–1624:1 (Neal Civjan).

[49] *See* Tr. 1356:3–9, ECF No. 1931 (Safra Catz).

[50] Tr. 887:23–889:3, 912:21–913:11, ECF No. 1909 (Rubin).

[51] Tr. 624:11–17, ECF No. 1907 (Rubin).

[52] Tr. 844:21–22, ECF No. 1909 (Rubin).

[53] "Feature phones" bridged the gap between pure cell phones, allowing only voice calls and text messaging, and smartphones. *Feature Phone*, PC Magazine Encyclopedia, https://www.pcmag.com/encyclopedia/term/feature-phone (last visited Feb. 7, 2020).

phones (Java ME and SE).[54] Indeed, prior to Android's launch, Java was "in over 85 percent of the [mobile phone] market."[55] Virtually every mobile phone on the market ran some version of Java.[56]

Moreover, even putting aside the fact that Java was already in use in smartphones, Sun and Oracle always expected that the Java platform would expand its presence in mobile devices as the processing power of these devices advanced. The Java platform was designed for all devices possessing a threshold level of processing power. Early mobile phones and other resource constrained devices ran Java ME—a slimmed-down derivative of Java SE. Illustrating Java's adaptation to advancing device capabilities, some early smartphones, such as the first Sidekick (which had a black-and-white screen, a handful of applications, and a primitive web browser) could run Java SE.[57]

Modern smartphones have processing power, graphics, applications, and browsers more advanced than the PCs that first ran Java. Smartphones today—many of which run Android—have a fully integrated email and messaging system, the speed and complexity of which makes mid-2000s PC and phone versions look quaint. The smartphone is no different in processing power from the PCs that were available

---

[54] By 2005, Java ME was already in about 79% of wireless handsets. Trial Ex. 134 at 3.

[55] Tr. 1624:21–24, ECF No. 1932 (Civjan).

[56] Trial Ex. 134 at 3.

[57] Klint Finley, *Tech Time Warp of the Week: Before the iPhone, Anyone Who Was Anyone Rocked a Sidekick*, Wired (May 22, 2015), https://www.wired.com/2015/05/tech-time-warp-week-iphone-anyone-anyone-rocked-sidekick/ (last visited Feb. 7, 2020).

when Android launched. The only difference is that smartphones today fulfill Java's long-standing goal of bringing extensive computing power and digital awareness to the masses in handheld devices.

In short, Java was always intended to be device and platform-independent. Google's use of Java in smartphones simply was not transformative.

V. GOOGLE'S USE UNDERMINES JAVA'S PROMISE OF "WRITE ONCE, RUN ANYWHERE"

Google's unauthorized use of Java has severely harmed the market for Java. This is in large part due to the way Google used the Java copyrighted material.

For one thing, Android was designed intentionally so that it would be *incompatible* with the JVM, effectively cutting Oracle out of the potential Android market.[58] Specifically, although Google took Java's declaring code verbatim for certain packages, it added other packages and omitted many others. This ensured that programs written for Android could not be run on any other platforms or devices and 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

---

[58] *See* Tr. 1332:1–2, ECF No. 987 (John C. Mitchell) ("So you don't really have compatibility. You can't ship code from one platform to another."); Tr. 1440:25–1441:2, ECF No. 1931 (Edward Screven) (Android's programming "locks programmers into Android . . . their applications can't run in other environments").

actions had two consequences: quick access to Java developers while preventing Java cross-platform compatibility. This undermined Java's vision of cross-platform compatibility. Because of Google's (largely uncommunicated) changes, 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."

Moreover, Google's business model of giving away Android source code for free, combined with "locking in" Java developers, effectively destroyed Java's licensing model.[59] Sun's Neal Civjan testified that Android "hijacked" Java: Google "took our technology and they gave it away for free and they took our customers and it was devastating."[60] For instance, Motorola eventually dropped its license for Java in favor of Google's free Android.[61]

Ultimately, Java found itself unable to compete with a free version of its own product. Former licensees were able to obtain for free, from Google, the same packages performing the same functions on the same platform for which they previously had to pay. As a result, Oracle eventually had to exit the smartphone

---

[59] Android makes money through advertising and other embedded search functions, while Sun/Oracle makes revenue from Java through commercial licensing fees paid by those seeking to use the Java platform in their devices and programs, as noted *supra*. Tr. 1771:17–23, ECF No. 1932 (Adam Jaffe).

[60] *Id*. at 1641:5–17 (Civjan).

[61] Tr. 1356:3–9, ECF No. 1931 (Catz).

market entirely, falling from a dominant position to a non-existent position in less than a decade.[62]

This was not simply the result of market forces at work. This was Google's calculated strategy to subvert the Java platform's market position by taking Oracle's technology, giving it away for free, and generating a revenue stream in a different manner such that no rational customer would remain with Oracle. Had Google created its own technology, its actions might be cutthroat but effective business. By stealing Oracle's longstanding copyrighted material, however, Google unfairly destroyed the market for Java.

And by doing so, Google has reopened 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.

[62] *Id*. at 1361:23–1362:2 (Catz).

CONCLUSION

For the foregoing reasons, *Amicus* respectfully submits that the Court should affirm the decisions of the Federal Circuit.

Respectfully submitted,

CELESTE L.M. KOELEVELD
Counsel of Record
JOHN P. ALEXANDER
YOUNG JUN CHOI
CLIFFORD CHANCE US LLP
31 W. 52nd St.
New York, New York 10019
(212) 878-8000
Celeste.Koeleveld@
CliffordChance.com

Counsel for Amicus Curiae

February 19, 2020