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 FOR PROFESSOR EUGENE H. SPAFFORD, PH.D., PROFESSOR ZHI DING, PH.D., PROFESSOR EMERITUS LEE A. HOLLAAR, PH.D., PROFESSOR ADAM PORTER, PH.D., AND MR. PETER KENT, BSC AS *AMICI CURIAE* IN SUPPORT OF RESPONDENT

WEIL, GOTSHAL
& MANGES LLP
767 Fifth Avenue
New York, NY 10153
(212) 833-3000

ANNE M. CAPPELLA
Counsel of Record
CHRISTOPHER M. PISTRITTO
WEIL, GOTSHAL & MANGES LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000
anne.cappella@weil.com
christopher.pistritto@weil.com

Counsel for Amici Curiae

February 19, 2020

294664

COUNSEL PRESS
(800) 274-3321 • (800) 359-6859

TABLE OF CONTENTS

TABLE OF CITED AUTHORITIES

INTEREST OF *AMICI CURIAE*[1]

Dr. Eugene H. Spafford is a Professor of Computer Science at Purdue University, where he has been employed since 1987, and the founder and Executive Director Emeritus of the Center for Education and Research in Information Assurance and Security at Purdue. He has over 30 years of experience in both practice and research in the field of computing and computer science, including with the use of application programming interfaces ("APIs"). Over the past decade, he has served in an advisory or consulting capacity on issues in computing and information systems with several U.S. government agencies and their contractors, including the National Science Foundation, the Federal Bureau of Investigation, the Government Accountability Office, the National Security Agency, the U.S. Department of Justice, the U.S. Air Force, the U.S. Naval Academy, the Department of Energy, and the Executive Office of the President. He has served on the President's Information Technology Advisory Committee and has testified before Congressional committees nine times. He is a Fellow of five major scientific and professional organizations involved with computing: the Association for Computing Machinery (ACM), American Academy for the Advancement of Science (AAAS), Institute of Electrical and Electronic Engineers ("IEEE"), International Information Systems Security Certifications Consortium, and Information Systems Security Association (ISSA).

1. Pursuant to Supreme Court Rule 37.6, counsel for *amici curiae* states that no counsel for a party authored this brief in whole or in part. No party or party's counsel contributed money that was intended to fund preparing or submitting this brief, and no person, other than the *amici* or *amici* counsel, made such a contribution. Petitioner has lodged a blanket amicus consent letter with the Court, and Respondent has consented to the filing of this brief.

Dr. Spafford has published and spoken extensively about software engineering, information security, and professional ethics, and he has served on the editorial boards of several major journals of computer science. He was affiliated with the Software Engineering Research Center, an NSF University-Industry Cooperative Research Center when it was located at Purdue. His current research is directed towards the architecture, construction, and public policy of secure information systems. He has been writing computer programs since 1972, including computer security programs that have been used internationally by government agencies and companies, and his programming experience includes Java and other programming languages. He also has taught undergraduate and graduate courses involving software engineering, information system security, and many programming languages.

Dr. Zhi Ding is a Professor of Electrical and Computer Engineering at the University of California, Davis. He has over 29 years of practical and research experience in the field of electronic and electrical engineering, including with the use of APIs. He received his Ph.D. degree in Electrical Engineering from Cornell University in 1990. He was a faculty member of Auburn University and the University of Iowa, and he has held visiting positions at Australian National University, Hong Kong University of Science and Technology, NASA Research Center (Cleveland, Ohio), and USAF Wright Laboratory.

Dr. Ding has published extensively about electrical engineering, and his research focuses on communications and systems. He has published over 200 journal papers and more than 230 conference papers. Dr. Ding is a coauthor of the popular engineering textbook Modern Digital and Analog Communication Systems (5th ed.). Dr. Ding is an IEEE Fellow, and he has served on the technical committees of several workshops and conferences. He was the Technical Program Chair of the 2006 IEEE Global Telecommunication Conference and the General Chair of the 2016 IEEE Conference on Acoustics, Speech, and Signal Processing.

Dr. Lee A. Hollaar is an Emeritus Professor at the School of Computing at the University of Utah. Before his retirement in 2014 after 34 years on the faculty, he taught courses in computer and intellectual property law and computer systems and networking. He has been programming computers since 1964 and designing computer hardware since 1969. He received his B.S. degree in electrical engineering from the Illinois Institute of Technology in 1969 and his Ph.D. in computer science from the University of Illinois at Urbana-Champaign in 1975.

Dr. Hollaar was a Fellow with the Senate Committee on the Judiciary and technical advisor to its chair, Senator Hatch, where he helped the Committee with the No Electronic Theft Act and Digital Millennium Copyright Act, as well as a number of other bills. He has been a Special Master in a number of software copyright cases in the Federal District Courts for the Eastern District of Michigan and the District of Puerto Rico, where he offered opinions and made recommendations on the scope of copyright protection in software and whether there was infringement. His amicus brief to the Supreme Court in *MGM v. Grokster* provided the inducement theory that became the basis of the Court's unanimous decision.

Dr. Hollaar is the author of Legal Protection of Digital Information (Second Edition 2016, online at Bloomberg Law).

Dr. Adam Porter is a Professor of Software Engineering at the University of Maryland and the University of Maryland Institute for Advanced Computing Studies. Dr. Porter was appointed as the Executive Director of the Fraunhofer Center for Experimental Software Engineering in July 2015. The Fraunhofer Center is a UMD-affiliated applied research center focusing on software that increasingly underlies most innovation. Dr. Porter earned his Ph.D. from the University of California at Irvine.

Dr. Porter is a Senior Member of the IEEE and a Senior Member of ACM, and served on the editorial boards of the IEEE Transactions on Software Engineering and the ACM Transactions on Software Engineering and Methodology. Dr. Porter has published extensively about software engineering, including the paper, Empirically Guided Software Development Using Metric-Based Classification, which was listed as one of the 20 most widely-cited articles published by IEEE Software. Dr. Porter has also taught a number of courses in software engineering, including in large scale software development. Dr. Porter also co-created a Massive Open Online course on Android development that has reached over 900,000 students.

Mr. Peter Kent is a best-selling author of dozens of technical books for both computer professionals and users, including multiple programming works such as *The Official Netscape JavaScript Book* and the *Official*

Netscape JavaScript Programmer's Reference over his 30 year publishing career. As an educator and trainer, Mr. Kent has taught a multitude of courses to over 30,000 students spanning the digital and high-tech realms. He has also served as an expert witness in dozens of legal cases directed to patent, copyright, trademark, and contract disputes concerning Internet technology and digital marketing.

Amici's interest in this appeal is to ensure a robust and balanced intellectual property regime that promotes innovation, reliability, and security in software and information systems. *Amici* have no interest in any party to this litigation or stake in the outcome of this appeal.

SUMMARY OF ARGUMENT

As professors, researchers, practitioners and authors in computer science, *amici* created, used, and taught others about software application program interfaces ("APIs")—including APIs used in the software that we have written, the research that we have overseen, the companies and government agencies that we have advised, and the courses that we have taught. Software APIs are widely used in our modern information systems. The public policy and legal treatment of APIs including under the Copyright Act, as well as what constitutes fair use of those APIs, is therefore of great academic and practical interest to us and others in the computer science, computer engineering, systems engineering, electrical engineering, and software engineering communities.

Amici agree with Oracle that APIs can be copyright protectable code, the copied Java APIs contain such copyrightable code, and Google's taking of the Java APIs was not a fair use. *Amici* further concur with the Federal Circuit's analysis in its prior rulings as they pertain to the Java APIs and Google's use thereof in its Android platform.

Amici agree that there is a potential public benefit in the open use of some, though not all, software interfaces, such as furthering interoperability. There are channels available for such use that do not violate copyright law. Consistent with those channels, Oracle offered three different types of licenses for Java. Yet Google did not take a license, and despite its copying did not make its Android platform interoperable with Java. *Amici* observe that this case should be judged – and our opinions are based on – current copyright and fair use law as they exist today, within the framework Congress has given.

1. The Federal Circuit correctly determined that the Java APIs have sufficient expression to be protectable by copyright. In particular, *amici* agree with the Court's finding that the copied Java APIs could have been written any number of ways while still achieving the same function and that the structure, sequence, and organization of those Java APIs are creative and original.

There are many types of APIs, and those that exhibit creativity in the design of their sequence, structure and organization or the declaring source code in the API are within the scope of source code protected by copyright. APIs can be expressed in many different ways yet still accomplish the same purpose and objective. Differences among APIs are often the result of subjective choices based on experience and experimentation, rather than dictates of functionality. Certain APIs are better than others designed for the same function.

The copied Java APIs are an example of such copyrightable source code. The ingenuity and creativity of its developers resulted in Java's well-designed, intuitive set of APIs spanning thousands of lines of code.

While a single line of source code from an API can and often does demonstrate creativity, to understand the creativity of the Java APIs, however, a court should look at the work as a whole. That Google and its *amici* choose to focus on only a few of the eleven thousand lines of copied API source code so as to decry a lack of creativity is both erroneous and tangential to the copyright analysis, which must look at the whole work that was taken. The design choices for the copied Java APIs along with the overall structure, sequence, and organization ("SSO") of those APIs involve a far greater degree of choice and expression, and amply demonstrate the creativity of its developers.

Even if interoperability was relevant to copyright protection, it is not at issue here. Google's copying of thousands of lines of Java API source code and the corresponding SSO under the guise of interoperability is misleading. *Amici* agree that widespread interoperability increases the efficiency of developers and the industry at large. Yet respondent does not seek to preclude interoperability by enforcing its copyrights. Software developers and academics can freely use the Java APIs to write applications on the Java platform for essentially any reason. However, if an entity wishes to copy the Java APIs *to develop a competing platform*, Oracle offers a license under certain terms. Indeed, Oracle offers a free open-source license called OpenJDK in exchange for the developer making its revisions public and free for others to use. Such contributions to the Java community enhance the value and interoperability of Java for all. Google refused to accept Oracle's licenses.

Nor was it necessary for Google, a company with vast resources, to copy portions of Java's API at all for its Android operating system. Other APIs existed at the time and there is no indication that Google choose Java for any reason other than its popularity with programmers. Moreover, Google's copying of Java API source code was not driven by technical necessity to achieve interoperability. Google was not seeking to allow Java applications to operate with the Android operating system, but instead to use the popularity of Java's APIs to attract programmers to its competing platform.

Google argues that to afford copyright protection to popular APIs would impede efforts to promote interoperability, and thus harm the industry overall. *Amici* acknowledge that interoperability between programs and platforms is often a desirable objective. Google's argument, however, is based on a false premise. The Android operating system was designed not to interoperate with Java applications that use Oracle's Java APIs. The Android internal product literature and Google's own witnesses stated as much in this case. A167; A21181:4-7; A21503:16-A21504:2; A22397:11-A22398:3; A22463:13-22. Applications using the Java programming language are not interoperable on the Android operating system. Likewise, applications for Android are not interoperable with the Java Platform Standard Edition ("Java SE") API. This is because Google did not copy *all* of Oracle's Java APIs, and as a practical matter a Java program needs one or more libraries, packages, or classes using those APIs that do not exist in Android or vice versa. Android is simply a competing software platform that included core features of the highly popular Java APIs to reduce the learning curve and make it more attractive for programmers.

2. The Federal Circuit then correctly determined that Google's copying of the Java APIs was not a fair use. The Federal Circuit considered each of the fair use factors in coming to its conclusions, with which *amici* agree, that Google's copying of the Oracle Java APIs was non-transformative and commercial, the 37 API packages at issue involved some level of creativity, and Google's conduct is the type that would likely result in an adverse impact on the potential market for original APIs.

Google's wholesale copying of Java API source code to attract programmers to its competing Android platform was not a fair use. The copied API source code was used for the same purpose, the APIs had the same expression and meaning as they did in Java SE, and Google's argument that applying Java APIs to smartphones constitutes a new context is meritless. The copied Java API source code was not "transformed" merely because it was re-implemented on a different type of computer (a smartphone rather than a server or desktop computer). The meaning and message of the source code in the copied Java APIs are intentionally identical to the original function. Even if this was transformative, Java existed in mobile phones prior to Android.

As to the second fair use factor, the creativity involved in developing the Java APIs as the Federal Circuit found indicates that the nature of the copyrighted work is one deserving of copyright protection. Even though copied APIs provide a set of functions, those functions can be expressed in multiple ways and designers exercise vast amounts of creativity in creating them.

In considering the third fair use factor, there is no doubt that Google copied far more of the Java APIs than necessary. Google stipulated that only 170 lines of source code were necessary to write in the Java language, yet Google copied over 11,000 lines of source code.

Regarding the fourth and final fair use factor of the effect upon the potential market, the Federal Circuit correctly stated that copying substantial portions of the Java APIs is precisely the sort of behavior that can, and as the record shows did, impact the market for the Java API. Failure to protect investments in APIs will likely have a similar deleterious effect on investments in the development of complex APIs. To the extent Google argues this sort of unauthorized copying is necessary for success in the industry, concrete counterexamples abound. Notably, Apple's iPhone has been an undeniable triumph in the same market as Google's Android platform without copying or even supporting Java. Indeed, Apple developed an entire language and API for programmers to use with its iPhone, and have met with great success. Moreover, programmers frequently learn new programming APIs, and can do so quickly (as they did with the Apple iOS platform) when the API is well designed.

Because well-designed APIs play a central role in software and information systems, it is important that the law recognize and protect the creativity they embody.

ARGUMENT

I. API STRUCTURES

A. APIs And Software APIs

To assist the Court in understanding the technology of the present case, we provide the following discussion of software APIs.

Modern, complex computer programs such as an operating system are typically structured in multiple pieces, with a main program that communicates with various other program components that perform different functions. Each program component comprises source code that performs said functions (e.g., storing data in memory). These program components also need to communicate with each other to access the functionality each provides. User applications such as a word processor, web browser, or game also need to access the functionality in these program components.

APIs simplify access to the functions in the program components, as developers using an API do not need to write or even know the source code in the program components themselves. Instead, developers simply need to know how to request the desired functions using the API. In essence, the developers can treat an API as a "black box" that receives an input and returns a result or output.

Generally speaking, an "application programming interface" or API "consists of the methods and variables programmers use to work with a component or tool in their applications."[2] In other words, APIs define how computer programs communicate with each other and the format of those communication. APIs come in a variety of types, forms, and levels of complexity, but all describe and enable interactions between various types of software.

A "software API" is an API used in a computer program. A software API is a computer component, which is written in source code and compiled like other computer source code. It defines how to communicate with a program to perform predefined functions and specifies the results that will be output by the program. For example, an API could express a particular method that draws a circle and defines the inputs (two coordinates and a radius measurement) and outputs (displaying a circle on screen).

Software developers have choices when designing an API. Designing the declaring code included in an API takes innovation and creativity to balance the competing tensions of functionality, flexibility, and simplicity, so it is attractive to application program developers who may have a choice of platforms. API developers must also anticipate the needs of the users of the API, now and in the future. In doing so, API designers not only select what functions the API will offer to application program developers and how those functions will be presented, but must design the APIs so they are expandable in the future. Creativity extends far beyond the individual methods: the

2. Patrick Niemeyer & Daniel Leuck, Learning Java 917 (4th ed. 2013).

API designers also strive to organize the API elements in an intuitive manner for application program developers. An API's final design thus represents only one of a vast number of possible approaches, and expresses the author's creativity and affirmative choices to perform select functions.

B. Java APIs

The Java Platform Standard Edition ("Java SE") allows an application program developer to access functions or methods in a hierarchy of nested data structures. At the highest level are libraries, which are each made up of a group of packages (e.g. security, language, math, input and output, etc.), each of which is made up of a group of related classes (e.g. the security package contains various classes directed to algorithms, authorization, encryption keys, etc.). A class in turn is a group of functions (in Java parlance a function is referred to as a "method"), each of which perform discrete tasks.

The fundamental unit of a Java API is a method. Each method includes two source code components. The *declaring code* is the source code at the start of a method, with the designer choosing a method name, various variables, and what inputs and outputs that method will perform when a developer requests it. The source code that implements the requested method, referred to as *implementing code,* runs in the developer's program.

Significant creativity goes into the design of the structure, sequence, and organization of the API itself, including how to structure the libraries, packages, classes, and methods, as well as the declaring code itself. Indeed, part of the beauty of Java is that groupings of methods and classes often share or "inherit" features that are commonly used, such that the decision of how to group classes and methods becomes a creative design choice, not just a categorizing or filing exercise.

The Java APIs allow developers to access the methods for use in their own program or application. Thus, instead of writing every line of code necessary to run a program from scratch, which she is free to do, a developer can use a method already defined in a Java API.

A method's declaring code can be lengthy and complex, such as the example below from Oracle's opposition to Google's petition for Certiorari.[3] The implementing code is elided for the sake of brevity):

```
public abstract void verify (PublicKey key, String
                    sigProvider)
      throws CertificateException, NoSuch-
             AlgorithmException,
             InvalidKeyException,
             NoSuchProviderException,
             SignatureException
             {. . .}
```

The Java declaring code itself thus illustrates that there is creativity in writing methods.

The structure, sequence and organization of Java methods and classes requires creativity as well. Methods in the same class and classes in the same package can and

3. Oracle's Br. In Opp. to Pet. for Cert. at 6.

often do share common functions and features. A method using these common functions eliminates the need to duplicate source code, simplifying the expression of that method. The Java API designer thus chooses how methods and classes are organized, how components interact, and what functions are shared for a desired function. Examining a single method in a class overlooks the structure, sequence and organization of the copied Java APIs.

These are only some of the many creative choices that go into writing API source code.

II. APIS OFTEN INVOLVE MANY CREATIVE CHOICES THAT ARE NOT DICTATED BY THE SOFTWARE'S FUNCTION

A. The Java APIs Are Creative And Not Dictated By Their Functions

Much like different words, sentence structures and literary styles can convey the same concepts, the source code used to express an API reflects the author's imagination and creativity rather than rigid dictates of pre-determined functions. Indeed, authoring elegant and intuitive API declaring code often involves even greater creativity, talent, experience, and subjective judgment than authoring the underlying implementing code.

A simplified example of the flexibility and variability in a Java API is instructive. Let us assume we desire an API that allows a user to draw different shapes. There are several ways to design a Java API to achieve this goal.

Figure 1 below shows one possible implementation, in which an API designer creates a "Polygon" class containing three methods for drawing three different shapes: “drawCircle,” “drawEllipse,” and “drawSquare.” An API designer named the class, decided its contents in the form of three Java methods, wrote the declaring code of each method, which includes choosing the inputs and outputs, and then wrote the implementation source code within the method.

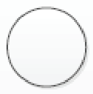
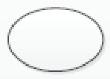
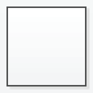
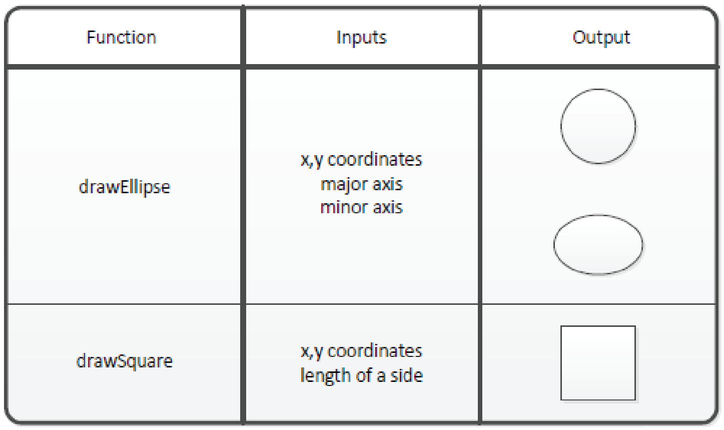
Function	Inputs	Output
drawCircle	x,y coordinates radius	
drawEllipse	x,y coordinates major axis minor axis	
drawSquare	x,y coordinates length of a side	

Figure 1.

Alternatively, the API designer may choose to design the class differently and name it “FlexibleShapeDrawing”. As shown in Figure 2 below, instead of two separate “drawCircle” and “drawEllipse” methods, this API includes a single “drawEllipse” method. This approach may be more intricate to use and implement because the user must now input two equal values for the major and minor axes to draw a circle. However, this design has the advantage of greater flexibility and power as it allows the user to draw multiple shapes with a single method.



Function	Inputs	Output
drawEllipse	x,y coordinates major axis minor axis	
drawSquare	x,y coordinates length of a side	

Figure 2.

As an even more flexible alternative, instead of separate methods for separate shapes, the API designer may instead include only the method "drawPolygon" in the "Polygon" class. As seen in Figure 3 below, "drawPolgyon" includes an additional "ShapeType" variable beyond variables for coordinates. The "ShapeType" variable is not defined in the "Polygon" class, but in an entirely new class within the same Java package. The class defining the complex data object "ShapeType" includes variables for the number of sides (*e.g.*, "0" for a circle or "4" for a square) and, for any given shape, the relevant size measurements (*e.g.*, the radius in the case of a circle or the length of each side in the case of a square). While this method may require more sophisticated knowledge to use and implement than those so far discussed, it can draw more complex shapes (*e.g.*, a pentagon).

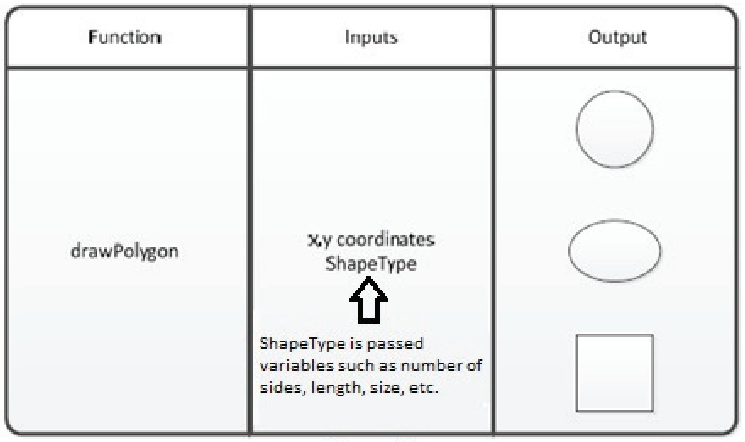

Function
Inputs
Output
drawPolygon
x,y coordinates
ShapeType
ShapeType is passed variables such as number of sides, length, size, etc.

Figure 3.

In each of these three designs an API designer had the same goal, namely to allow a programmer to draw shapes. Each API designer chose a different way to access shape drawing functionality, how complex the code should be, the design, and the names, arguments, and implementations of the methods.

Of course, these are only a few possible API designs for a handful of methods, and importantly, there is no "right" answer. There are many more ways that an API designer may develop and structure an API to perform the same function. An API designer could have written drawCircle, drawEllipse, and drawSquare into separate classes instead of one class, renamed them freely, designed a separate class or package dedicated to drawing images that each method referenced or called, and many more possibilities.

An API designer must make similar decisions for each and every method and class in an API. For complex APIs involving hundreds or thousands of methods and classes, the permutations of expressive choices are potentially exponential, resulting in a vast number of ways API designs can be expressed. The many possible approaches to the simple shape-drawing example illustrates the myriad creative choices that an API designer must make, including balancing factors such as functionality, flexibility, and simplicity.

Designing APIs is analogous, for example, to the innumerable creative choices that an author makes in writing a story. The author must consider various narrative elements, such as the characters, locations, and plot points, and the relationships and specific details of each. She must also consider how to describe and arrange these elements into the sentences, paragraphs, chapters, and so forth to create not only the order and structure of the story, but how that story is expressed. Authoring most APIs requires similar decisions regarding the interactions an API facilitates, accessible functions, implementation details, and internal structures and relationships. While the most simplistic and purely functional APIs may lack creativity, this is not the case with Java SE.

B. Android Is Not Interoperable With Java

The slogan "write once, run anywhere" coined by the creators of the Java programming language described the "core value proposition of the Java platform."[4] As a classic learning text on Java explained, "this means that the most important promise of Java technology is that you have to write your applications only once – for the Java platform – and then you'll be able to run it *anywhere*."[5]

4. David Flanagan, Java In A Nutshell at 4-5 (5th ed. 2005).

5. *Id.*

Android did not adopt this policy. Android cannot run all Java programs. A2205 ("Does Android support existing Java apps? A. No. . . . Is Android Java compatible? A. No."). We understand this was affirmed by the evidence and witness testimony presented during trial. Pet. App. 268a (noting Android's creation led to "fragmentation" with Java); Transcript of Jury Trial Proceedings, at 1010:4-7 (Doc. 967), at 1331:16-1332:2 (Doc. 987), at 2221:11-2222:3 (Doc. 1065), at 2287:13-22 (Doc. 1065). Because Google selected only certain Java SE API packages, the vast majority of Java programs (relying on Java API packages that do not exist in Android) are not interoperable. Such existing Java programs must be modified or rewritten from scratch to run on Android. The reverse is also true, as the great majority of Java programs that rely on the Android API will not operate with Java SE.

Based on our understanding as computer science professors and professionals there was no need from a technical perspective for Google to copy the declaring code and SSO for a subset of Java APIs. Copying was not necessary to create a platform that was interoperable with Java SE, a platform using the Java language, or a new platform at all.

Instead, Google copied 37 packages from the Java APIs because many programmers used Java. "Because Java is a simple and elegant language with a well-designed, intuitive set of APIs," "programmers write better code with fewer bugs than for other platforms, thus reducing development time."[6] Rather than go through the effort and expense of developing a robust set of APIs, Google instead

6. *Id* at 7.

chose simply to copy Oracle's Java APIs to attract Java programmers to write applications for Android. Writing code for Android comes at the expense of writing code for Java SE since Android programs are not interoperable with Java SE.

C. Failing To Protect APIs Would Harm The Industry

Google argues that allowing copyright protection over Java APIs would have deleterious effects on the industry. Instead, it would have the opposite effect. Should APIs be stripped of copyright protection, the incentive for an individual or company to invest in the lengthy development of robust and ingenious new APIs would be lessened. Sun Microsystems spent thirteen years developing Java prior to Oracle's purchase of Sun for over $5.5 billion dollars.[7] Oracle has since released seven new versions of Java SE over the last decade.[8] Without copyright protection, anyone could have copied the Java SE source code or any API wholesale, released a commercial competing product without incurring any development costs whatsoever, and there would be no recourse.

7. Press Release, Oracle Corp., *Oracle Buys Sun* (Apr. 20, 2009), https://www.oracle.com/corporate/pressrelease/oracle-buys-sun-042009.html.

8. *Oracle Java Archive*, Oracle Corp., https://www.oracle.com/java/technologies/oracle-java-archive-downloads.html (last visited Feb. 13, 2020)

III. GOOGLE'S COPYING OF THE JAVA APIS WAS NOT A FAIR USE

A. The Declaring Code And SSO Of APIs Embody Substantial Creativity

We understand that one of the factors in the fair use analysis is "the nature of the copyrighted work," 17 U.S. Code § 107, and that this involves an analysis of the creativity of the underlying copied work. In this case, we understand that Google does not dispute that it copied the declaring code of 37 Java API packages and the associated "structure, sequence, and organization" ("SSO"). The copied portions include classes, methods, and variables contained in the Java API, their organization and relationships, how to use them, and their expected behavior.

As explained in §I.A, designing the declaring code for an API involves substantial creativity and a wide range of choices. APIs can be expressed in many ways yet still accomplish the same purpose and objective, and as such any differences among APIs are often because of subjective choices based on experience and experimentation rather than dictates of functionality.

As depicted in the examples provided in §II.A, there are creative design choices at each level of an API, which can lead to APIs with significantly different expressions in the form of source code and SSOs to achieve the same goals. An API designer has to decide which methods, classes, packages, and other elements to develop. The API designer must then decide how to express these elements, including their behavior, complexity, and relationships with the program components. Classes and packages can be rearranged, interfaces can be implemented in one API but not in another, and ultimately the overall structure of APIs that perform the same functions may be different. The expression of the declaring code is part of the API design: for example, how to name the methods, the selection and ordering of inputs and outputs, and the types of errors that are reported. Each step in the design process leaves room for the imagination and independent judgment of the API author.

That the resulting set of source code from this creative process performs a function does not undermine the creativity involved in its design. The expression of an API is not dictated by function. Certain APIs are better than others designed for the same function. Accordingly, a court considering this factor in the fair use inquiry should find that APIs, like the copied Java APIs, are creative in nature.

B. The Declaring Code And SSO Are Significant Portions Of The Creative Work Of An API

As discussed above, we understand that another factor in the fair use analysis is the amount and substantiality of the portion used in relation to the copyrighted work as a whole, which includes a qualitative determination.

As discussed in §II.A, the declaring code and SSO of an API embodies a qualitatively significant portion of the creativity involved in authoring an API. In some instances, the declaring code and SSO of an API might comprise most of the creative work of the API. Also as discussed in this section, the selection and ordering of the packages, classes, and methods may have nearly uncountable options, as may the naming and selection of the declaring code. The implementing code, in contrast, may be relatively straightforward or even limited by the confines of the creative decisions made in designing the declaring code and SSO. For such APIs, the creativity embodied in the declaring code and SSO exceeds that for the implementing code, and the declaring code and SSO comprise most of the creative work involved in the authorship of the API.

The "Polygon" example in §II.A amply demonstrates how the declaring code and SSO can embody a relatively greater portion of the creative work of an API than the implementing code. There, the author of each of the three Java API examples found a different, creative way to draw shapes. Another dozen API designers may have chosen a dozen different designs. Once those expressive design choices for the declaring code and SSO have been made, the implementation of the method(s) potentially may involve comparatively less creativity and more rote implementation work. The source code to draw a circle or square is mundane and straightforward in comparison to the choice of how to design the drawing interface in the first place.

Finally, the declaring code and SSO are also significant portions of the expression of a software API because the application developers see and use these in writing programs. To learn how to write programs using the software API, for example, a developer could consult an API specification. The API specification identifies the libraries, packages, classes, and methods that are available for use by the developer, their declaring code, and their respective intended functions and relationships. The expression of the software API available ("exposed") to developers is the declaring code and SSO of the API. Developers need not view or even be aware of the implementing code, which can be treated as a black box, to successfully learn and use the API.

C. The Creativity Of An API Is Not Substantively Changed By Substituting New Implementing Code For The API's Existing Implementing Code

We understand that another fair use factor is "the purpose and character of the use," which has been interpreted by courts to include a determination of whether the use is transformative, and further that a transformative use is one that changes the expressive content, meaning, or message of the underlying work. We also understand that it is undisputed in this case that, in most instances, Google used different implementing code, but retained the declaring code and SSO, for the 37 Java API packages it copied.

We understand that Android's implementing code performed substantially the same function as the Java implementing code for the copied packages. For example, the Android implementing code accepts the same parameters and generates the same return values and exceptions as the Java implementing code. Indeed, we understand that Google's implementing code replicated the exact same Java functionality, contained the same SSO of the packages, classes, and methods, and the declaring code that was described in the Java API specification.

The declaring code and SSO of the copied Java API packages in Android thus retained the same purpose, function, and meaning each had in Java SE. The use of different implementing code in Android did not substantively change or add to the purpose or creative expression in the copied declaring code and SSO. In other words, the underlying expression of the declaring code and SSO was not transformed.

D. Google Did Not Transform The Java APIs When Copying Them Onto A Smartphone Platform

We also understand that the determination of whether a work is transformative examines whether the work uses the copyrighted material for a different or distinct purpose. Here, Google argues that placing Java on the Android mobile smartphone platform is transformative.[9] Yet from the standpoint of a software API, there is no significant difference between these processor-based platforms, because a smartphone is ultimately simply a general purpose computer that is small enough to fit in a person's hand. This is especially the case for an API designed to work across platforms, such as Java SE. Indeed, some recent tablets and smartphones contain processors utilizing the same architecture found in full-sized computers, and now many laptop computers use touchscreens that are functionally equivalent to those used on smartphones.[10] There are no appreciable differences

9. Google's Pet. for Cert. at 25; *see also* Google Br. at 43.

10. *See, e.g.*, *Surface Pro Specifications*, Microsoft Corp. https://www.microsoft.com/surface/en-us/support/surface-pro-specs (last updated Apr. 12, 2019) (tablet containing Intel x64 architecture processor, which has been used in desktops and

with respect to the copied Java APIs whether on a server, a desktop computer, or a smartphone.

Furthermore, the purpose of, and creative expression for, software APIs for an application programming platform is the same regardless of the size of the device on which it is running: Its purpose is to describe the syntax, functions, variables, and data structures that a programmer can learn and use.

E. There Is No Technical Need To Copy The Declaring Code And SSO Of The Java APIs

Based on our understanding as computer science professors and professionals, there was no need from a technical or technological perspective to copy the declaring code and SSO for a subset of Java APIs, as Google did in this case. Googled admits that the "declarations were not beyond Google's capacity to create."[11] Rather, Google chose to use certain Java APIs to attract programmers instead of developing their own or choosing another available set of APIs. In light of its decision to use the Java SE API in Android, Google now claims that copying the declaring code and SSO for the API was necessary. There was no technical need to use the Java SE API and therefore no need to copy the declaring code and SSO of the Java SE API.

In the District Court proceedings, we understand that the parties stipulated that only 170 lines out of more than 11,000 lines of code Google copied were necessary if

laptops).

11. Google Br. at 14.

one wanted to use the Java programming language.[12] The additional lines of API source code and the SSO appear to have been copied to attract Java application programmers to a different platform by offering them the ability to use some of their knowledge of Java, rather than learning a completely new platform. Further, as discussed above in §II.B, we understand that Google conceded Android is not interoperable with Java. In other words, Google did not copy the Java APIs out of technological necessity to allow programs designed for Java SE to operate on Android.

Google was not pressed for choices in choosing which language and APIs to use for its Android platform. Java was one of many programming languages with accompanying APIs such as Python (whose creator worked at Google while Android was being developed),[13] Go or Golang (invented at Google while Android was being developed),[14] C++, Ruby, Perl, C#, and many more. None of these are proprietary, and as such Google could have used any one.

---

12. *Oracle Am., Inc. v. Google LLC*, 886 F.3d 1179, 1206 (Fed. Cir. 2018) (stating, "[o]n remand, the parties stipulated that only 170 lines of code were necessary to write in the Java language. It is undisputed, however, that Google copied 11,500 lines of code—11,330 more lines than necessary to write in Java. That Google copied more than necessary weighs against fair use.").

13. Guido van Rossum, *Resume*, Github, https://gvanrossum.github.io/Resume.html (last visited Feb. 13, 2020).

14. Rob Pike, *Go at Google: Language Design in the Service of Software Engineering*, GoLang, https://talks.golang.org/2012/splash.article (last visited Feb. 13, 2020).

Indeed, given its expertise with software, Google could also have followed Apple's example and developed its own set of APIs for Android developers to use. Apple developed the Swift programming language and APIs specifically for use on Apple iOS products including its iPhone platform.[15] While the development of Swift took Apple four years from initial development to its first release, and no doubt many engineers and much expense, the iPhone has been an inarguable success.[16]

In sum, Google did not choose Java for interoperability, nor because it was pressed by a lack of available APIs or could not develop its own APIs. Instead, Google chose Java APIs because Java was popular and using those APIs could attract programmers to develop application for its platform. "The final, and perhaps most important reason to use Java is that programmers like it."[17] This rationale for copying cannot be the basis of a fair use.

15. *Apple Releases iOS 8 SDK With Over 4,000 New APIs*, Apple Inc. (June 2, 2014), https://www.apple.com/newsroom/2014/06/02Apple-Releases-iOS-8-SDK-With-Over-4-000-New-APIs/

16. Tripp Mickle, *Apple Posts Record Revenue on Strong iPhone, App Sales*, Wall St. J. (Jan. 28, 2020), https://www.wsj.com/articles/apple-posts-revenue-growth-on-strong-airpod-app-sales-11580247318 ("Sales of iPhones, which account for more than half of its revenue, rose 8% to $55.96 billion").

17. David Flanagan, Java In A Nutshell at 6 (5th ed. 2005).

F. Google's Commercial Use Of The Java APIs Without A License Materially Impaired The Market Value

We understand that one of the factors in the fair use analysis is the effect of the use upon the potential market for the work. Accordingly, an additional consideration is that Oracle had provided means by which Google could have "fairly" used Oracle's Java code that would have preserved the market that Oracle had intended and created for Java as an "open source" project.

Oracle offered three different licenses for Java SE. *Oracle Am., Inc. v. Google Inc.*, 750 F.3d 1339, 1350 (Fed. Cir. 2014). Because Oracle had an existing licensing scheme, there was an available avenue for Google to use the Java code fairly. This included a *free* open source license called OpenJDK, which required only that the developer licensee make its revisions freely available to all.[18] Instead, Google copied the Java APIs without transforming the work, enabling interoperability, or contributing back to the Java community, which Oracle's licenses sought to establish, encourage, and protect. This severely undermined Oracle's rights to control the market for Java and its derivative works.

18. Oracle Br. In Op. to Pet. For Cert. at 7.

CONCLUSION

For the forgoing reasons, amici respectfully submit that the Court should affirm.

Respectfully submitted,

Weil, Gotshal
& Manges LLP
767 Fifth Avenue
New York, NY 10153
(212) 833-3000

Anne M. Cappella
Counsel of Record
Christopher M. Pistritto
Weil, Gotshal & Manges LLP
201 Redwood Shores Parkway
Redwood Shores, CA 94065
(650) 802-3000
anne.cappella@weil.com
christopher.pistritto@weil.com

Counsel for Amici Curiae

February 19, 2020