Oracle v. Google
Java API Copyrightability Put to the Test in Smartphone Technology Battle
By Barry Werbin

Software copyright battles resulting in precedential decisions excite technology-oriented copyright lawyers because of their infrequent issuance. But when a smartphone technology war is waged between two modern industry titans involving a case of first impression, as it was recently in Oracle America, Inc. v. Google, Inc., all eyes get raised, especially when the copyrightability of a core component of interoperative software development—application programming interfaces (APIs)—is thrown into question for the first time in the United States. In a case as notable for its substantive rulings on copyrightability of APIs as it was for its tripartite trial procedure, Oracle’s closely watched copyright and patent infringement action against Google, involving the ubiquitous Java programming language, ended with an opinion holding that portions of the APIs in issue were not protectable by copyright because they were barred under the idea-expression merger doctrine or as a functional system of operation, and otherwise constituted unprotectable short names or phrases. But a widely anticipated ruling on the copyrightability of APIs generally did not materialize, leaving that question to be assessed on a case-by-case basis under the guidelines established by the court (should the decision be upheld on appeal).

This article primarily addresses the court’s ultimate conclusion rendered in its 47-page opinion on May 31, 2012, that, as a matter of law, the particular attributes Google utilized from the structure, sequence, and organization of 37 Java API packages in issue are not protectable by copyright under the 1976 Copyright Act. To be understood is the court’s decision, a rudimentary lesson in the Java programming language (as “taught” by District Court Judge William Alsup) and an overview of the history of copyrightability of computer programs in the United States are instructive.

Java Programming Language—Background
Sun Microsystems, Inc., was the original developer of the Java programming language. Since its introduction in 1996, more than 9 million software developers have used Java, which now powers not only numerous Internet web applications, but more than 3 billion mobile phone systems, including Google’s popular Android system. By 2007, Sun had re-released Java as free open source code under a GNU General Public License. In 2010, Oracle Corporation acquired Sun and its Java programming language, renaming Sun “Oracle America, Inc.” (Oracle). Later that same year, Oracle sued Google for patent and copyright infringement arising out of Google’s Android operating system, which incorporates elements of the Java API. While Java was and remains a free open source platform, Oracle continued to license the Java API to third-party developers (as had Sun). While Google did not obtain such a license, it asserted that it used free and open technology to develop its own Android system using Java, and that such use of Java was a “fair use” and not infringing.

One of the primary advantages that has led Java to become “one of the world’s most popular programming languages and platforms” is its ability, through the use of a “virtual machine,” to allow programmers to write applications that can run on different types of computer hardware without the need to rewrite code for each type. In order for different software programs to communicate with each other, it is necessary to provide APIs, which act like a set of traffic control mechanisms that control each program’s input and output, including specific functions. The prewritten programs that comprise an API are known as “class libraries.” As described by Judge Alsup:

An API is like a library. Each package is like a bookshelf in the library. Each class is like a book on the shelf. Each method is like a how-to-do-it chapter in a book. Go to the right shelf, select the right book, and open it to the chapter that covers the work you need.

As of 2008, the Java API (or class library) was comprised of 166 “packages” or “folders,” which in turn were broken out into more than 600 prewritten programs or “classes,” comprising over 6,000 methods (or “subroutines”) of operation or instruction. Oracle accused Google of infringing 37 of the Java API packages by copying their exact names and functions, including the “structure, sequence and organization of the overall code for the 37 API packages.” There was no dispute, however, that Google had not copied the literal software code contained in the API packages themselves.

Oracle further accused Google’s mobile Android operating system, including Google’s own Android virtual machine called “Dalvik,” of infringing certain patents covering the Java system. While seven Java-related patents originally were in suit, four were invalidated by the United States Patent and
Dissecting the Java API Structure

To understand the court’s ruling on non-copyrightability of the API elements used by Google, it is essential to understand and dissect the various components and structure of the API packages themselves, as explained by Judge Alsup, who provided an overview of Java API language programming. To begin, the relevant Java code uses a series of syntax separators (e.g., {, }, ;), operators (e.g., +, -, *, /, <, >), literal values (e.g., 123, ‘x’, ‘Foo’) and keywords (e.g., if, else, while, return). Additionally, also used are “identifiers (e.g., String, java.lang.Object), which are used to name specific values, fields, methods, and classes. . . .”

These syntax elements, in turn, form statements, which “are commands that tell the computer to do work” and are written in specific sequences. A key component of the API programming structure are “methods” (or “declarations”), which are a form of subroutines; methods or subroutines dictate specific procedures or functions. In programming parlance, a method is associated with a “class,” which is a prewritten program, and Java classes are combined into class libraries. If a method also has a return type, it requires a statement, which includes a line of code to be returned when that part of the program is reached.

A method consists of the method header and the method body. A method header contains the name of the method; the number, order, type and name of the parameters used by the method; the type of value returned by the method; the checked exceptions that the method can throw; and various method modifiers that provide additional information about the method. Critical to the court’s rulings was that the header line of code in a method introduces the body of the method, specifying the inputs, name, and functionality of the method. Thus, any programmer who wants to provide a method with the same functionality must write the header code “in the same way and must do so no matter how different the implementation may be from someone else’s implementation.”
Within the method body is coding (also called an “implementation”) that implements the particular method, which does “the heavy lifting, namely the actual work of taking the inputs, crunching them, and returning an answer.”

Google did not copy the actual Java code for the API method bodies but created its own implementations, which accounted “for 97 percent of the code for the 37 packages.” Once created, a method responds to a “call” emanating from code located elsewhere, such as in a different program, which calls on or invokes the method for whatever function it is designed specifically to perform. As noted, one or more methods then comprise a “class” and a given Java program consists of one or more classes, which, in turn, comprise “packages,” into which all Java code must be placed. Also included in classes are fields, which hold values and methods that operate on those values. A class “declaration” (or “header”) is a line that includes the name of the particular class and other information defining that class. Classes can have subclasses that “inherit” the functionality of the class itself, thereby eliminating the need for programmers to rewrite portions of the inherited code each time a new subclass is written; a declaration alerts the compiler that “the fields and methods of the parent class are inherited automatically into the new subclass. . . .”

Because Java restricts subclasses to their original parent class, inheriting (or “extending”) fields and methods from other parent classes would not normally be possible. To resolve this, Java uses “interfaces” (not to be confused with APIs), which have their own source code files but act like a class that can be inherited by other classes. Classes and interfaces are grouped into “packages,” similar to how files are organized into folders on computers.

Judge Alsup emphasized that “declaration” lines were “entirely dictated by the rules of the [Java] language. In order to declare a particular functionality, the language demands that the method declaration take a particular form. There is no choice in how to express it.” Although declarations must be the same for the same functionality, “the names of the methods and the way in which the methods are grouped do not have to be the same.” But while the same functionality could be accomplished using different application interfaces, they would not be interoperable because “code written for one API would not run on an API organized differently, for the name structure itself dictates the precise form of command to call up any given method.” As Judge Alsup noted, the whole point of the “tried-and-true pre-packaged programs in the [Java] API” is to avoid a programmer’s need to reinvent the wheel, as the APIs are “ready-made to perform a vast menu of functions. . . . Users and developers can supplement the API with their own specialized methods and classes.”

Java API Elements Found to Be Uncopyrightable

Against this important technical background, the copyright claims came down to two distinct issues: (1) whether the Java API method names and headers were protected; and (2) whether the “structure, sequence and organization” of the API packages were protected, to the extent Google “generally replicated the overall name organization and functionality of 37 packages in the Java API.” Put more specifically, the second issue assessed “whether Google was and remains free to group its methods in the same way as in Java, that is, to organize its Android methods under the same class and package scheme as in Java.”

As to the first issue, Judge Alsup held that because “names, titles and short phrases” are not protectable under copyright, it follows that the method names and headers are such equivalents and thus not entitled to protection. This holds true, where, as here, “the Android method and class names could have been different from the names of their counterparts in Java and still have worked. . . .”

With respect to the second issue, which Judge Alsup described as “the more difficult question,” the order concluded that the specific aspects of the Java API structure, sequence and organization that Google utilized constituted a “system or method of operation” that was not protected by copyright under § 102(b) of the 1976 Copyright Act, which provides that “[i]n no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.” Judge Alsup further concluded that “the taxonomy [of the subject APIs] is composed entirely of a system of commands to carry out specified computer functions” and is therefore similarly barred under § 102(b). As he noted, “[e]veryone agrees that no one can copy line-for-line someone else’s copyrighted computer program. When the line-by-line listings are different, however, some copyright owners have nonetheless accused others of stealing the ‘structure, sequence and organization’ of the copyrighted work. That is the claim here.”

To support his rulings, and because this was a case of first impression addressing the copyrightability of APIs, Judge Alsup provided a lengthy, detailed chronology of the relevant decisional law to date addressing the copyrightability of computer programs and their particular attributes, including “structure, sequence and organization.” While a complete survey of that case law is beyond the scope of this article, a succinct history follows for critical context.

A Concise History of Computer Program Copyright Protection

The 1909 Copyright Act did not contain any specific protections for software, which became a subject of ongoing debate; however, the Copyright Office began accepting applications to register software in May 1964 as “books.” Under the 1976 Copyright Act, software was originally included within the statutory definition of “literary works” under § 101. The United States formally granted express copyright protection to “computer programs” as part of the Computer Software Copyright Act of 1980, which added a definition of “computer program” in § 101 of the United States Copyright Act. The level of copyright protection afforded software has gone through various iterations, with the consensus that both original source code, written in “conventional human language,” and binary machine-readable or object code, qualify for protection generally as “literary works” and “computer programs” under the definitions set forth in § 101 of the Copyright Act. Beyond copying a program’s actual coding, referred to
as the “literal” elements, a program’s overall organization and arrangement may also be protected under copyright as “non-literal” elements. But because coding itself is comprised of various components and modules that serve different purposes and functions, the analysis of the types of code, and related components and structure, that are entitled to protection can be complex, which is evident from key prior decisions that Judge Alsup reviewed as part of his analysis. In particular, much attention has been paid to the Copyright Act’s absolute exclusion of protection under § 102(b) for any “idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.”

Germaine to any analysis of nonliteral elements of software is the Supreme Court’s seminal decision in Feist Publications, Inc. v. Rural Telephone Service Co., involving telephone white pages. In Feist, the court held that a compilation of nonprotectable factual elements could still be entitled to copyright protection even if its constituent elements are not, but expressly rejected the “sweat of the brow” theory. As Judge Alsup emphasized, the court made it clear that such protection is “thin,” and anyone is “free to use the facts contained in another’s publication to aid in preparing a competing work, so long as the competing work does not feature the same selection and arrangement.”

Judge Alsup’s starting point of analysis is the Supreme Court’s 1879 decision in Baker v. Selden, which, despite ongoing criticism over the decades, has not been overruled. Obviously predating any concept of computer code, Baker involved a method of bookkeeping utilizing a novel blank form, where an alleged infringing form did not copy actual textual material from original but embodied the same system used by the plaintiff with a different arrangement of columns and headings. Ruling that only patent law can grant exclusive rights to methods, the court also established the “merger” doctrine, where systems and methods are conjoined with text and diagrams illustrating them. In essence, the court ruled that copyright protection cannot apply to methods and diagrams where they are “incident” to the art to be practiced (sounding much like patent law), lest a monopoly be granted on the underlying art itself.

While nothing in the 1976 Copyright Act expressly alludes to Baker, the Act nevertheless confirms the basic proposition that copyright may not be claimed in “any idea, procedure, process, system, method of operation, concept, principle or discovery.” But because the 1976 Act (which became effective January 1, 1978) also continued pre-1978 law with respect to rights in “useful articles,” it has been left largely to the courts to apply these criteria to computer software, an assessment of which Judge Alsup undertook in significant detail. To truncate his analysis, the charts on the previous pages summarize the key circuit court cases relevant to software and computer applications, as discussed by Judge Alsup in his order.

**Judge Alsup’s Key Findings of Non-Copyrightability**

While noting that the case law has become “more cautious” and deferential to § 102(b), Judge Alsup emphasized that infringement of “structure, sequence and organization” was not a “dead letter” and the Computer Associates approach has become the trend. Following his lengthy decisional analysis, Judge Alsup summarized the key principles governing the API claims in the case as follows:

- Under the merger doctrine, when there is only one (or only a few) ways to express something, then no one can claim ownership of such expression by copyright.
- Under the names doctrine, names and short phrases are not copyrightable.
- Under § 102(b), copyright protection never extends to any idea, procedure, process, system, method of operation, or concept regardless of its form. Functional elements essential for interoperability are not copyrightable.
- Under Feist, we should not yield to the temptation to find copyrightability merely to reward an investment made in a body of intellectual property.

The court reiterated the main issue as whether Google violated the Copyright Act when it copied the “overall name organization and functionality of the 37 packages in the Java API” and whether such elements were copyrightable. With respect to the method level of the API, Judge Alsup emphasized that as long as the method implementation code differs, anyone is free to write a method that mimics the same function or specification of any of the Java API methods. Rejecting Oracle’s claim of ownership in any manner of implementation of its methods, the court held that others are “free to write their own implementation to accomplish the identical function [because] ideas, concepts and functions cannot be monopolized by copyright.”

Despite evidence by Oracle that the design of its methods and the structure of its classes were creative endeavors, which Judge Alsup acknowledged, such functionality was only protected under the Patent Act. Citing Computer Associates, Judge Alsup held that a method specification is an unprotected “idea” and a method implementation is protected expression, because “no matter how creative or imaginative a Java method or specification may be, the entire world is entitled to use the same method specification (inputs, outputs, parameters) so long as the line-by-line implementations are different.” Moreover, under Java rules, the method specification must be identical to carry out the same function. Because of this need to be identical, the merger doctrine “bars anyone from claiming exclusive copyright ownership of that expression.” Accordingly, Google was found not to have infringed any copyrights at the method level “where the heavy lifting is done.”

With respect to the class structures, Java rules require naming classes and using “strict syntax and punctuation in the lines of code that declare a class.” While the name of a particular class can be different, names alone are not protected by copyright; similarly, the declaration line must be identical to provide the same functionality and also is not protectable.

As for whether Google was permitted to organize its Android methods under the same class and package scheme as Java, Oracle argued that Java does not require that its methods or classes be grouped in any particular order and, therefore, its “overall system of organized names—covering 37 packages,
with over six hundred classes, with over six thousand methods” was entitled to protection as a “taxonomy” under the Seventh Circuit’s decision in *American Dental Association v. Delta Dental Plans Association*, which involved the ADA’s “Code on Dental Procedures and Nomenclature.”[78] Judge Alsup’s response was that while the Java program structure “resembles a taxonomy,” it also is a command structure for a system or method of operation within the Java API and, as such, is not protectable under § 102(b) of the Copyright Act, which denies protection for a “system” or “method of operation,” even where, as with Java, the system or method “has thousands of commands arranged in a creative taxonomy.”[79]

Further distinguishing the Java command structure from the ADA’s taxonomy, Judge Alsup emphasized the need for “interoperability” between Google’s Android system and pre-existing Java code, which required “the same java.package. Class.method() command system using the same names with the same ‘taxonomy’ and with the same functional specifications. Google replicated what was necessary to achieve a degree of interoperability—but no more, taking care, as said before, to provide its own implementations.”[80] The Ninth Circuit’s opinions in *Sega* and *Sony* also were cited as supporting the principle that interface procedures required for interoperability are “functional requirements for compatibility” and are thus not copyrightable under § 102(b) of the Copyright Act.[81] In an analogy to the game console interoperability issues in those two cases, Judge Alsup observed that “Google was free to duplicate the command structure for the 37 packages in Android in order to accommodate third-party source code relying on the 37 packages (taking care to write its own implementations).”[82]

Thus, the taxonomy in *American Dental*—a process coding system to be used for billing and insurance claims purposes—was deemed not to be analogous because it did not address the far more complex issues involving computer code structure and itself was not a system of commands. Moreover, Judge Alsup noted that the ADA’s taxonomy did not come with “instructions,” whereas the Java API was bundled with documentation and embedded comments that “describe every package, class and method, what inputs they need, and

<table>
<thead>
<tr>
<th>NON-9TH CIRCUIT LAW</th>
<th>ESSENTIAL HOLDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc., 797 F.2d 1222 (3d Cir. 1986)</td>
<td>Program software written in EDL programming language was rewritten by the defendant line by line in BASIC language to compete with the plaintiff’s application. The court held where there is nonliteral copying of a program’s file structure, the copyright idea/expression dichotomy focuses on “the end sought to be achieved by the work in question. [T]he purpose or function of a utilitarian work would be the work’s idea, and everything that is not necessary to that purpose or function would be part of the expression of the idea.” The structure of the subject program was deemed copyrightable because there were many other ways to perform the same function using different structures and designs.[41]</td>
</tr>
<tr>
<td>Computer Associates International, Inc. v. Altai, 982 F.2d 693 (2d Cir. 1992)</td>
<td>Leading case often cited, devised three-part “abstraction-filtration-comparison” test, under which nonliteral parts of a program are dissected into structural parts, separating out components in the public domain, incorporated ideas, and expression incidental to such ideas. Structures are then filtered out if dictated by efficiency or external factors (such as a computer’s design standards), or found in the public domain. Finally, the remaining components are compared for substantial similarity. Criticized Whelan for its narrow view of a program’s “idea.”[42]</td>
</tr>
<tr>
<td>Gates Rubber Co. v. Bando Chemical Industries, Ltd., 9 F.3d 823 (10th Cir. 1993)</td>
<td>Agrees with Altai filtering approach, emphasizing external factors and the need to “eliminate the unprotectable elements of processes, facts, public domain information, merger material, scènes à faire material, and other unprotectable elements suggested by the particular facts of the program under examination.”[43] Defined a program’s structure as a “description of how the program operates in terms of its various functions, which are performed by discrete modules, and how each of these modules interact with each other.”[44] Under § 102(b) of the Copyright Act, unprotected “processes” can be found “as part of the system architecture, as operations within modules, or as algorithms.”[45]</td>
</tr>
<tr>
<td>Lotus Development Corp. v. Borland International, Inc., 49 F.3d 807 (1st Cir. 1995), aff’d per curium, 516 U.S. 233 (1996) (4–4 evenly divided court)</td>
<td>Lotus 1-2-3 spreadsheet program commands were arranged in multiple menus and submenus. In its competing spreadsheet program, Borland copied the Lotus 1-2-3 menu hierarchy to make it compatible to enable Lotus customers to easily switch to Borland. No underlying code was copied. The court held Lotus’s menu hierarchy (as opposed to Lotus’s actual code) was not copyrightable because it was a method of operation under § 102(b) that was necessary for use of the program’s functional capabilities.[46]</td>
</tr>
<tr>
<td>Hutchns v. Zoll Medical Corp., 492 F.3d 1377 (Fed. Cir. 2007)</td>
<td>Applying Lotus, a plaintiff’s claim of copyright protection in a CPR program’s “system of logic whereby CPR instructions are provided by computerized display” was rejected because copyright does not protect “technologic method of treating victims by using CPR and instructing how to use CPR.”[47]</td>
</tr>
</tbody>
</table>
what outputs they return—the classic form of instructions for use,” which further emphasized the existence of a “system” or “method of operation” under § 102(b).83

Finally, the court made the important point that, at least in the Ninth Circuit, the extent to which the structure, sequence, and organization of a computer program is entitled to protection will vary depending on the particular facts of each case.84 In concluding his extensive opinion, Judge Alsup seemed perturbed at the breadth of Oracle’s claims, a theme which appears several times in the opinion, by observing that “[t]o accept Oracle’s claim would be to allow anyone to copyright one version of code to carry out a system of commands and thereby bar all others from writing their own different versions to carry out all or part of the same commands. No holding has ever endorsed such a sweeping proposition.”85

The European Union Rules on Software Functionality and Copyrightability

Coincidentally, on May 2, 2012, the European Court of Justice (ECJ) came down with its own significant and analogous ruling.

Oracle v. Google

The case has overtones of Oracle v. Google and an analogy to the Java API structure issues. SAS developed a database system, which contains a core component called “Base SAS” that enables users “to write and run their own application programs in order to adapt the SAS System to work with their data (scripts)”; these scripts “are written in a language which is peculiar to the SAS System.”86 Defendant World Programming did not have access to SAS’s source code and did not decompile its object code. As the ECJ noted,

The court established two-part extrinsic and intrinsic test for determining similarity between computer programs for purposes of assessing infringement, but not copyrightability. With respect to nonliteral features of a program, concepts that are fundamental to computer programs generally (ability to access files, edit, print, etc.) are non-copyrightable. As for functions listed in a menu bar, they constitute the idea of the program, while the expression of the ideas inherent in the program’s features was distinct. Lastly, pull-down windows are not protectable if they are standard or commonplace in the software industry, and in this case the defendant’s window looked different.87

Atari Games Corp. v. Nintendo of America Inc., 975 F.2d 832 (Fed. Cir. 1992) (based on Ninth Circuit law)

Atari reverse-engineered a Nintendo program that blocked game consoles from accepting unauthorized game cartridges, and wrote a new program in a different programming language that unlocked the consoles by generating the same signals as Nintendo’s program. A preliminary injunction based on copyright infringement was affirmed because the arrangement of Nintendo's program contained unique copyrightable expression and creative elements beyond a mere idea or process. This arrangement generated a data stream that did not “merge” with the process “so long as alternate expressions are available.”88

Table: 9th Circuit Law and Essential Holding

<table>
<thead>
<tr>
<th>9TH CIRCUIT LAW</th>
<th>ESSENTIAL HOLDING</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson Controls, Inc. v. Phoenix Control Sys., Inc., 886 F.2d 1173 (9th Cir. 1989)</td>
<td>At the preliminary injunction stage, the nonliteral structure, sequence, and organization of a program, which was customizable for each customer, was protectable under copyright where there was &quot;some discretion and opportunity for creativity in the structure,&quot; which was &quot;expression rather than an idea.&quot;86</td>
</tr>
<tr>
<td>Brown Bag Software v. Symantec Corp., 960 F.2d 1465 (9th Cir. 1992)</td>
<td>The court established two-part extrinsic and intrinsic test for determining similarity between computer programs for purposes of assessing infringement, but not copyrightability. With respect to nonliteral features of a program, concepts that are fundamental to computer programs generally (ability to access files, edit, print, etc.) are non-copyrightable. As for functions listed in a menu bar, they constitute the idea of the program, while the expression of the ideas inherent in the program’s features was distinct. Lastly, pull-down windows are not protectable if they are standard or commonplace in the software industry, and in this case the defendant’s window looked different.87</td>
</tr>
<tr>
<td>Atari Games Corp. v. Nintendo of America Inc., 975 F.2d 832 (Fed. Cir. 1992) (based on Ninth Circuit law)</td>
<td>Atari reverse-engineered a Nintendo program that blocked game consoles from accepting unauthorized game cartridges, and wrote a new program in a different programming language that unlocked the consoles by generating the same signals as Nintendo’s program. A preliminary injunction based on copyright infringement was affirmed because the arrangement of Nintendo's program contained unique copyrightable expression and creative elements beyond a mere idea or process. This arrangement generated a data stream that did not “merge” with the process “so long as alternate expressions are available.”88</td>
</tr>
<tr>
<td>Sega Enterprises Ltd. v. Accolade, Inc., 977 F.2d 1510 (9th Cir. 1992)</td>
<td>Accolade copied Sega's object code that was used as an interface to make game cartridges compatible with Sega's game consoles, then wrote its own interface source code to copy the interface procedure to allow its cartridges to play in Sega consoles, where Sega's unlock &quot;key&quot; consisted of only 20 bytes of initialization code plus the letters &quot;S–E–G–A.&quot; Copying of the object code was held to be fair use because &quot;the interface procedures for compatibility were functional aspects not copyrightable under Section 102(b).&quot;89 Atari was distinguished &quot;because there was only one signal that unlocked the Sega console, unlike the 'multitude of different ways to unlock' the Nintendo console….&quot;90</td>
</tr>
<tr>
<td>Sony Computer Entertainment, Inc. v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000)</td>
<td>An emulator allowed Sony PlayStation console games to be played on desktop computers. Connectix duplicated the interface procedures of the consoles by copying the object code in the BIOS system of the PlayStation operating software in order to ascertain the signals sent between the BIOS and console. It then wrote its own code to implement that interface. Copying the object code for this purpose constituted fair use under § 102(b) and Sega because the BIOS contained unprotected functional elements.91</td>
</tr>
</tbody>
</table>
“[b]y means of observing, studying and testing the behaviour of SAS Institute’s program, WPL reproduced the functionality of that program by using the same programming language and the same format of data files.”

The ECJ emphasized that “the main advantage of protecting computer programs by copyright is that such protection covers only the individual expression of the work and thus leaves other authors the desired latitude to create similar or even identical programs provided that they refrain from copying.” In this case, however, the ECJ ruled that “neither the functionality of a computer program nor the programming language and the format of data files used in a computer program in order to exploit certain of its functions constitute a form of expression of that program and, as such, are not protected by copyright in computer programs. . . .” Nevertheless, the ECJ also held that the reproduction, either in a computer program or a manual for such program, of elements described in a user manual for a different program that is protected by copyright under national law (in this case that of the United Kingdom), may infringe the copyright in the protected manual if under such national law “that reproduction constitutes the expression of the intellectual creation of the author of the user manual for the computer program protected by copyright.”

The Future of Copyright Protection for Software Structure, Sequence, and Organization

Despite Judge Alsup’s cautionary note that his rulings do not vitiate copyright in all APIs generally, the noose appears to be tightening both under U.S. and EU law with respect to replication of the structure, sequence, and organization of computer programs, where the implementation—i.e., the original expression—of software functionality is accomplished through independently created code. While the Java API packages themselves provided a road map for application of the idea/expression and merger doctrines embodied in § 102(b) of the Copyright Act, the decision sheds light and provides important guidance on the complexities and nuances inherent in assessing the copyrightability of software where nonliteral copying is in issue. Altai’s often-cited “abstract-filtration-comparison” test continues to bar protection for a program’s structure where similarities are due to external factors and efficiencies. Now the Ninth Circuit will have the task of reconciling its 1989 decision in Johnson Controls, Inc., which acknowledged copyrightability of a program’s structure, sequence, and organization where it qualifies as an “expression” of an idea and not the idea itself, with its later opinions in the Sega and Sony cases, where software interfaces necessary for interoperability created a fair use defense. At the same time, EU member state courts now have valuable guidance from the ECJ on limiting protections for a program’s functionality that is replicated in another program.

Endnotes


2. Judge Alsup had previously defined an API as “what allows software programs to communicate with one another. It is a set of definitions governing how the services of a particular program can be called upon, including what types of input the program must be given and what kind of output will be returned. APIs make it possible for programs (and programmers) to use the services of a given program without knowing how the service is performed. APIs also insulate programs from one another, making it possible to change the way a given program performs a service without disrupting other programs that use the service.” Oracle America, Inc. v. Google Inc., 810 F. Supp. 2d 1002, 1005 (N.D. Cal. 2011).


6. Computer code offered under a “free” open source license model is not stripped of copyright protection. See Jacobsen v. Katzer, 535 F.3d 1373, 1382 (Fed. Cir. 2008) (The defendant’s violations of an open source license for a “Java Model Railroad Interface” did not merely constitute a breach of contract, but copyright infringement. “The clear language of the [license] creates conditions to protect the economic rights at issue in the granting of a public license,” and that a violation of such license conditions constituted copyright infringement.).

7. Order at 4. The Java “virtual machine” is a runtime engine that interprets and executes Java coding on any computer, without being dependent on a specific hardware system or operating system. Essentially, it is software stored on a computer that runs applications written in Java. Definition of Java, PCMag.com, http://www.pcmag.com/encyclopedia_term0,1237,t=Java&i=45557,00.asp (last visited Oct. 25, 2012).

8. Order at 5.

9. Id. “In computer science, a subroutine, also termed procedure, function, routine, method, or subprogram, is a part of source code within a larger computer program that performs a specific task and is relatively independent of the remaining code.” See also Subroutine, WIKIPEDIA, http://en.wikipedia.org/wiki/Subroutine (last visited Oct. 25, 2012).


11. The ’104 Patent describes a “method and apparatus for resolving data references in generated code” and the claim focused on whether Google’s source code utilized what is called “symbolic references,” which relate to how data is tagged when software is compiled, i.e., converted from readable computer coding (source code) to computer-only readable binary code that executes commands (object code). The ’520 Patent describes a “method and system for performing static initialization,” which relates to ways in which different classes of files can be consolidated using less code in virtual machines using what is called “simulated execution.” See Caleb Garling, Google and Oracle Enter Round Two of Heavyweight Legal Fight, WIRED (May 8, 2012, 5:34 PM), http://www.wired.com/wiredenterprise/2012/05/google-oracle-patent-phase (last visited Oct. 25, 2012).


13. Id. at 13.

14. Id. at 14.

15. Id. at 13, 14. Programmers write software as source code, which is readable by humans and reflects statements and instructions as text (words and phrases), numbers and symbols. In order to be read and understood by a computer device, source code must be converted or “compiled” into object or machine code, which is a binary code of ones and zeros only computers can read and interpret. Id. at 5; see Source Code, TECHTERMS.COM, http://www.techterms.com/definition/
sourcecode (last visited Oct. 25, 2012). Java requires an intermediate type of source code conversion into what is known as “bytecode,” which in turn is converted into binary machine or object code by the Java virtual machine. Order at 3.

16. Order at 14 n.1. With respect to these eight decompiled bytecode files and the nine lines of rangeCheck code, Judge Alsup observed:

Line by line, Oracle tested all fifteen million lines of code in Android (and all files used to test along the way leading up to the final Android) and these minor items were the only items copied, save and except for the declarations and calls which, as stated, can only be written in one way to achieve the specified functionality.

Id. at 14. The referenced “declarations and calls” were at the heart of the API dispute and are discussed infra.


20. Id. at 8.


22. Order at 8.

23. Id.

24. Id.

25. Id. at 8–9.

26. Id. at 9.

27. Id.

28. Id.

29. Id.

30. Id. at 9–10.

31. Id. at 10.

32. Id. The court illustrated this concept with a hypothetical API:

package java.lang; // Declares package java.lang
public class Math { // Declares class Math
    public static int max (int x, int y)
        { // Declares method max
            if (x > y) return x ; // Implementation, returns x or else return y ; // Implementation, returns y
        } // Closes method
    } // Closes class

The first line defines the package, the second defines the class, and the third defines the specific method or function. To invoke the above method from another program or class, the following “call” (or statement) would be included in that program: “int a = java.lang.Math.max (2, 3).” This statement instructs the computer to find the “java.lang” package, then the “max” method within the “Math” class, “input ‘2’ and ‘3’ as arguments, and then return a ‘3’, which would then be set as the value of ‘a.’” The designation “public” in the above illustration means that any program can call on it. Similarly, each of the specified parameters above “is drawn from a short menu of possibilities, each possibility corresponding to a very specific functionality.” The markers used (such as “[]”) also are mandatory. Id. at 10–11.


34. Id. at 34.

35. Id. at 36.

36. Id. at 14–15, also noting this preclusion was adopted by the Ninth Circuit in Sega Enterprises, Ltd. v. Accolade, Inc., 977 F.2d 1510, 1524 n.7 (9th Cir. 1992). The court also cited to Copyright Office Circular 34: “Copyright law does not protect names, titles, or short phrases or expressions. Even if a name, title, or short phrase is novel or distinctive or lends itself to a play on words, it cannot be protected by copyright.” Information Circulars and Fact Sheets, U.S. COPYRIGHT OFFICE, http://www.copyright.gov/circs (last visited Oct. 25, 2012).

37. Order at 3.

38. 17 U.S.C § 102(b); Order at 37–38.


40. Id. at 19.

41. Order at 20–21 (noting that Whelan has been criticized for its broad pronouncement of copyrightability based on a structure, sequence, and organization theory). See, e.g., Sega Enters., Ltd. v. Accolade, Inc., 977 F.2d 1510, 1524–25 (9th Cir. 1992).

42. Order at 21–23. See also Softel, Inc. v. Dragon Med. and Scientific Comm’ns, Inc., 118 F.3d 955 (2d Cir. 1997) (reaffirming the Altai test, and noting that the “the scene a faire [sic] doctrine denies protection to program elements that are dictated by external factors such as ‘the mechanical specifications of the computer on which a particular program is intended to run’ or ‘widely accepted programming practices within the computer industry.’”) 118 F.3d at 963 (quoting Altai)). The scènes à faire doctrine literally means “scenes that must be done” and in the context of copyright law refers to standard, stock, and common attributes that are not entitled to a copyright monopoly. In the context of computer software, this is akin to external factors that dictate specific conformity and functionality. See, e.g., Lexmark Int’l, Inc. v. Static Control Components, Inc., 387 F.3d 522, 535 (6th Cir. 2004).

43. Order at 23.

44. 9 F.3d at 835; Order at 23.

45. 9 F.3d at 837; Order at 23.

46. Order at 24–26. See also Broderbund v. Unison World, 648 F. Supp. 1127 (N.D. Cal. 1986), an early case that extended copyright protection to a menu structure (a greeting card design and printing program).

47. Id. at 26.


49. Id.; 17 U.S.C § 101.


51. 17 U.S.C. § 101 n.5 (“A ‘computer program’ is a set of statements or instructions to be used directly or indirectly in a computer to order bring about a certain result.”).


53. Order at 15–32.
57. Id. at 27–28.
58. Id. at 29.
59. Id. at 30. “Accolade copied Sega’s software solely in order to discover the functional requirements for compatibility with the Genesis console—aspects of Sega’s programs that are not protected by copyright. 17 U.S.C. § 102(b).” 977 F.2d at 1522.
60. Order at 30.
61. Id. at 32.
63. 101 U.S. 99 (1879).
64. Nimmer, supra note 48 at § 2.18[B][1]; Order at 15–16.
65. Order at 15–16.
66. Id. at 16; see Nimmer, supra note 48 at § 2.18[B][2] (“where use of the ‘art’, i.e., the idea, which a copyrighted work explains (or embodies) necessarily requires a copying of the work itself, then such copying will not constitute an infringement of copyright. However, if such copying occurs not in using the art but rather in explaining it, then such copying will constitute an infringement.”).
68. Nimmer, supra note 48 at § 2.18[B][4]; 17 U.S.C. § 113(b) (“This title does not afford, to the owner of the copyright in a work that portrays a useful article as such any greater or lesser rights with respect to the making, distribution, or display of the useful article so portrayed than those afforded to such works under the law, whether title 17 or the common law or statutes of a State, in effect on December 31, 1977, as held applicable and construed by a court in an action brought under this title.”).
69. Order at 32.
70. Order at 33–34.
71. Id. at 34.
72. Id.
73. Id. at 35.
74. Id.
75. Id. at 36. The court also observed that “all of the methods in the Java version can be found somewhere in the Android version, classified under the same classes.” Id. at 37 n.10. Google had also asserted a scènes à faire defense with respect to the organization of methods within a class generally. While such defense might have been applicable to certain specific methods, because evidence necessary to establish such a defense with respect to the hundreds of classes in issue was not presented at trial, Judge Alsup’s Order rejected “Google’s global argument based on scènes à faire.” Id. at 36–37 n.9.
76. Id. at 36.
77. Id.
78. 126 F.3d 977 (7th Cir. 1997); Order at 36–37. The American Dental Association (ADA) published the “Code on Dental Procedures and Nomenclature,” which classified all dental procedures into groups and assigned to each procedure a number and both long and short descriptions. Defendant’s work, entitled “Universal Coding and Nomenclature,” included most of the numbering system and short descriptions from the plaintiff’s work, but also made certain modifications to which the ADA objected because it upset the dental procedure coding convention it had established for uniformity. 126 F.3d at 977–78, 981. The Court of Appeals described a taxonomy as “a way of describing items in a body of knowledge or practice,” distinct from a compilation of facts. Id. at 980. The court held that this taxonomy—“numbers, short descriptions, and long descriptions”—was all copyrightable subject matter, noting: “Facts do not supply their own principles of organization. Classification is a creative endeavor [and] each scheme of classification could be expressed in multiple ways.” Id. at 979. Similarly, the taxonomy was held not be an uncopyrightable “system” under § 102(b) or otherwise barred under the idea/expression merger doctrine incorporated into that Section and first enunciated in Baker v. Selden. Id. at 980–81. Defendant’s modifications, which irked the ADA, also were held to be an unauthorized derivative work. Id. at 981.
79. Order at 37. The court also noted that Java dictates that commands specifying a particular function must be in this form: “java. package.Class.method(“).” See supra note 32.
80. Order at 38 (emphasis in original).
81. Id. at 39.
82. Id.
83. Id. at 40.
84. Id. The court relied on Johnson Controls for the point that the structure, sequence, and organization of a program is not automatically entitled to copyright protection but has to be assessed on a case-by-case basis. See Johnson Controls, 886 F.2d at 1175.
85. Order at 41.
88. Article 10(1) of the Agreement on Trade-Related Aspects of Intellectual Property Rights (TRIPs), to which the United State also is a party, provides: “Computer programs, whether in source or object code, shall be protected as literary works under the [Berne Convention].” This is subject to limitations in Article 9(2) of TRIPs that “ideas, procedures, methods of operation or mathematical concepts” are not protectable. Similarly, the ECJ also cited to the WIPO Copyright Treaty, dated December 20, 1996 (adopted in the EU on March 14, 2010, and in the U.S. on March 6, 2002), which made it clear that computer programs are to be treated as a form of “literary work” for purposes of copyright protection. The WIPO Treaty, which supple- ments the Berne Convention, provides in Article 4 that “[c]omputer programs are protected as literary works within the meaning of Article 2 of the Berne Convention. Such protection applies to computer programs, whatever may be the mode or form of their expression.” See Article 2 of the Berne Convention. (“The expression ‘literary and artistic works’ shall include every production in the literary, scientific and artistic domain, whatever may be the mode or form of its expres-sion...”) But Article 2 of the WIPO Treaty limits this right generally: “Copyright protection extends to expressions and not to ideas, procedures, methods of operation or mathematical concepts as such.”
89. SAS Inst., supra note 86, at ¶ 23.
90. Id. at ¶ 41.
91. Id.
92. Id. at ¶ 46.
93. Id. at ¶ 70.