Table of Contents

15

Software

> The programmer, like the poet, works only slightly removed from pure thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. Few media of creation are so flexible, so easy to polish and rework, so readily capable of realizing grand conceptual structures. Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.[1]

1. Frederick P. Brooks, *The Mythical Man-Month* (1975)

Software presents a subtly different functionality problem than physical designs do. Software is plainly functional: when run on a computer, it does something. But there are at least two distinct ways in which software might be more than just functional. First, the software's code might contain nonexpressive elements: multiple different programs can do the same thing, so a programmer typically has at least some design choices not dictated by a given function. Second, the thing the software does might be expressive: it might show a movie or play a song. (Recall *Stern*'s treatment of a video game as an audiovisual work.) Complicating matters even more, a program's output might itself be both aesthetic and functional, which is typically the case for user interfaces. Pay attention to how each body of intellectual property law teases out these different aspects of software.

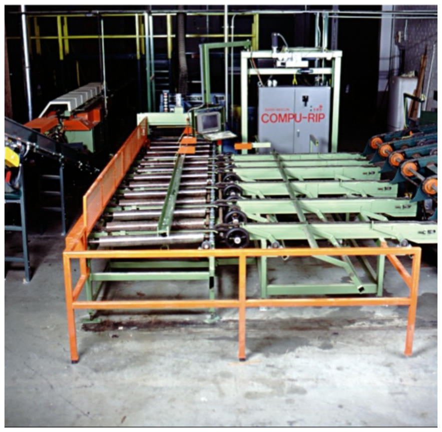
A Trade Secret

Trade secret protection is obviously available and effective for software used internally within a business. The more interesting question is whether and how a business can make software available to others while mainting at least some of the software's design as a secret. A later chapter will consider whether contractual restrictions help, but for now, focus on the practical question of what software necessarily discloses to its users.

Barr-Mullin, Inc. v. Browning
424 S.E.2d 226 (N.C. Ct. App. 1993)

According to defendants, the COMPU-RIP software is not a trade secret since it is (1) not subject to reasonable efforts to maintain its secrecy and (2) defendants reverse engineered this software. The COMPU-RIP software is contained in the form of ”programmable read-only memory chips” (PROMS) imbedded in the COMPU-RIP machinery. These PROMS contain only the ”object code” version of the computer program. This is the version of the computer software which is ”read” by the computer's machinery. Computer programmers do not write computer software in object code; rather, the software is written in ”source code” and then translated into object code so that the computer can execute the program. Since the COMPU-RIP software was sold in PROM form, the source code was not available to the general public.

The affidavits of Timothy Toombs and Gary Ruggles, who holds a Ph.D. in Electrical Engineering, indicate that because the COMPU-RIP software is distributed in object code form it is practically impossible to make any meaningful changes to the software. This evidence establishes the COMPU-RIP software was subject to reasonable efforts to maintain its secrecy. As to the question of reverse engineering, the affidavits indicate that it is practically impossible to make any modification to the COMPU-RIP software using only the object code contained in the PROMS. We find the evidence presented establishes the COMPU-RIP software was not ”readily ascertainable” through reverse engineering.



Barr-Mullin Compu-Rip rip saw feeder

B Patent

Alice Corp. v. CLS Bank Int'l
134 S. Ct. 2347 (2014)

The patents at issue in this case disclose a computer-implemented scheme for mitigating ”settlement risk” (i.e., the risk that only one party to a financial transaction will pay what it owes) by using a third-party intermediary. The question presented is whether these claims are patent eligible under 35 U.S.C. § 101, or are instead drawn to a patent-ineligible abstract idea. We hold that the claims at issue are drawn to the abstract idea of intermediated settlement, and that merely requiring generic computer implementation fails to transform that abstract idea into a patent-eligible invention.

I

A

Petitioner Alice Corporation is the assignee of several patents that disclose schemes to manage certain forms of financial risk. According to the specification largely shared by the patents, the invention ”enabl[es] the management of risk relating to specified, yet unknown, future events.”

The specification further explains that the "invention relates to methods and apparatus, including electrical computers and data processing systems applied to financial matters and risk management."

The claims at issue relate to a computerized scheme for mitigating "settlement risk" – i.e., the risk that only one party to an agreed-upon financial exchange will satisfy its obligation. In particular, the claims are designed to facilitate the exchange of financial obligations between two parties by using a computer system as a third-party intermediary. The intermediary creates "shadow" credit and debit records (i.e., account ledgers) that mirror the balances in the parties' real-world accounts at "exchange institutions" (e.g., banks). The intermediary updates the shadow records in real time as transactions are entered, allowing only those transactions for which the parties' updated shadow records indicate sufficient resources to satisfy their mutual obligations. At the end of the day, the intermediary instructs the relevant financial institutions to carry out the "permitted" transactions in accordance with the updated shadow records, thus mitigating the risk that only one party will perform the agreed-upon exchange.

In sum, the patents in suit claim (1) the foregoing method for exchanging obligations (the method claims), (2) a computer system configured to carry out the method for exchanging obligations (the system claims), and (3) a computer-readable medium containing program code for performing the method of exchanging obligations (the media claims). All of the claims are implemented using a computer; the system and media claims expressly recite a computer, and the parties have stipulated that the method claims require a computer as well.

II

Section 101 of the Patent Act defines the subject matter eligible for patent protection. We have long held that this provision contains an important implicit exception: Laws of nature, natural phenomena, and abstract ideas are not patentable.

We have described the concern that drives this exclusionary principle as one of pre-emption. See, e.g.,[2] (upholding the patent "would pre-empt use of this approach in all fields, and would effectively grant a monopoly over an abstract idea"). Laws of nature, natural phenomena, and abstract ideas are the basic tools of scientific and technological work. Monopolization of those tools through the grant of a patent might tend to impede innovation more than it would tend to promote it," thereby thwarting the primary object of the patent laws. We have repeatedly emphasized this concern that patent law not inhibit further discovery by improperly tying up the future use of these building blocks of human ingenuity.

2. Bilski v. Kappos ("*Bilski v. Kappos*"), 561 U.S. 593 (2010).

At the same time, we tread carefully in construing this exclusionary principle lest it swallow all of patent law. At some level, all inventions embody, use, reflect, rest upon, or apply laws of nature, natural phenomena, or abstract ideas. Thus, an invention is not rendered ineligible for patent simply because it involves an abstract concept. See[3] Applica-

3. Diamond v. Diehr, 450 U.S. 175 (1981).

tions of such concepts to a new and useful end remain eligible for patent protection.

Accordingly, in applying the S 101 exception, we must distinguish between patents that claim the building blocks of human ingenuity and those that integrate the building blocks into something more, thereby transforming them into a patent-eligible invention, The former would risk disproportionately tying up the use of the underlying ideas, and are therefore ineligible for patent protection. The latter pose no comparable risk of pre-emption, and therefore remain eligible for the monopoly granted under our patent laws.

III

In *Mayo Collaborative v. Prometheus Labs*, we set forth a framework for distinguishing patents that claim laws of nature, natural phenomena, and abstract ideas from those that claim patent-eligible applications of those concepts. First, we determine whether the claims at issue are directed to one of those patent-ineligible concepts. If so, we then ask, what else is there in the claims before us? To answer that question, we consider the elements of each claim both individually and as an ordered combination to determine whether the additional elements transform the nature of the claim into a patent-eligible application. We have described step two of this analysis as a search for an "inventive concept" – i.e., an element or combination of elements that is sufficient to ensure that the patent in practice amounts to significantly more than a patent upon the ineligible concept itself.

A

We must first determine whether the claims at issue are directed to a patent-ineligible concept. We conclude that they are: These claims are drawn to the abstract idea of intermediated settlement.

The "abstract ideas" category embodies the longstanding rule that an idea of itself is not patentable. In[4] for example, this Court rejected as ineligible patent claims involving an algorithm for converting binary-coded decimal numerals into pure binary form, holding that the claimed patent was "in practical effect a patent on the algorithm itself." And in[5] we held that a mathematical formula for computing "alarm limits" in a catalytic conversion process was also a patent-ineligible abstract idea.

4. Gottschalk v. Benson, 409 U. S. 63 (1972).

5. Parker v. Flook, 437 U.S. 584 (1978).

We most recently addressed the category of abstract ideas in *Bilski v. Kappos* ("*Bilski v. Kappos*"). The claims at issue in *Bilski v. Kappos* described a method for hedging against the financial risk of price fluctuations. Claim 1 recited a series of steps for hedging risk, including: (1) initiating a series of financial transactions between providers and consumers of a commodity; (2) identifying market participants that have a counterrisk for the same commodity; and (3) initiating a series of transactions between those market participants and the commodity provider to balance the risk position of the first series of consumer transactions. Claim 4 put the concept articulated in claim 1 into a simple mathematical formula. The remaining claims were drawn to examples of hedging in commodities and energy markets.

All members of the Court agreed that the patent at issue in *Bilski v. Kappos* claimed an abstract idea. Specifically, the claims described the basic concept of hedging, or protecting against risk. The Court explained that "hedging is a fundamental economic practice long prevalent in our system of commerce and taught in any introductory finance class." The concept of hedging as recited by the claims in suit was therefore a patent-ineligible abstract idea, just like the algorithms at issue in *Gottschalk v. Benson* and *Parker v. Flook*.

On their face, the claims before us are drawn to the concept of intermediated settlement, i.e., the use of a third party to mitigate settlement risk. Like the risk hedging in *Bilski v. Kappos*, the concept of intermediated settlement is a fundamental economic practice long prevalent in our system of commerce. The use of a third-party intermediary (or "clearing house") is also a building block of the modern economy. Thus, intermediated settlement, like hedging, is an "abstract idea" beyond the scope of § 101.

B

Because the claims at issue are directed to the abstract idea of intermediated settlement, we turn to the second step in *Mayo*'s framework. We conclude that the method claims, which merely require generic computer implementation, fail to transform that abstract idea into a patent-eligible invention.

1

At *Mayo* step two, we must examine the elements of the claim to determine whether it contains an "inventive concept" sufficient to "transform" the claimed abstract idea into a patent-eligible application.[6] A claim that recites an abstract idea must include "additional features" to ensure "that the claim is more than a drafting effort designed to monopolize the abstract idea."

6. "Perhaps the most striking feature of the opinion is its focus on the requirement for an 'inventive concept.' This choice of terminology is perhaps unfortunate, as it closely resembles the terminology of 'inventive step' that is used in much of the world to designate the requirement for a patentable invention to be a significant advance over the prior art. Typically the 'inventive step' requirement, which is found in many patent systems, including as Art. 52 of the European Patent Convention, is considered equivalent to the 'non-obviousness' requirement found in Sec. 103 of the American patent statute, rather than equivalent to any U.S. subject matter provision." Dan L. Burk, *The Inventive Concept in Alice Corp. v. CLS Bank Int'l.*, 45 IIC 865 (2014).

The introduction of a computer into the claims does not alter the analysis at *Mayo* step two. In *Gottschalk*, for example, we considered a patent that claimed an algorithm implemented on a general-purpose digital computer. Because the algorithm was an abstract idea, the claim had to supply a new and useful application of the idea in order to be patent eligible. But the computer implementation did not supply the necessary inventive concept; the process could be carried out in existing computers long in use. We accordingly held that simply implementing a mathematical principle on a physical machine, namely a computer, is not a patentable application of that principle.

Parker is to the same effect. There, we examined a computerized method for using a mathematical formula to adjust alarm limits for certain operating conditions (e.g., temperature and pressure) that could signal inefficiency or danger in a catalytic conversion process. Once again, the formula itself was an abstract idea and the computer implementation was purely conventional. In holding that the process was patent ineligible, we rejected the argument that implementing a principle in some specific fashion will automatically fall within the patentable subject matter of § 101. Thus, *Parker* stands for the proposition that the prohibition against patenting abstract ideas cannot be circumvented by attempting to limit the use of the idea to a particular technological environment.

In *Diamond v. Diehr*, by contrast, we held that a computer-implemented process for curing rubber was patent eligible, but not because it involved a computer. The claim employed a well-known mathematical equation, but it used that equation in a process designed to solve a technological problem in conventional industry practice. The invention in *Diamond* used a thermocouple to record constant temperature measurements inside the rubber mold – something the industry had not been able to obtain. The temperature measurements were then fed into a computer, which repeatedly recalculated the remaining cure time by using the mathematical equation. These additional steps transformed the process into an inventive application of the formula. In other words, the claims in *Diamond* were patent eligible because they improved an existing technological process, not because they were implemented on a computer.

These cases demonstrate that the mere recitation of a generic computer cannot transform a patent-ineligible abstract idea into a patent-eligible invention. Stating an abstract idea ″while adding the words ′apply it′″ is not enough for patent eligibility. *Mayo*. Nor is limiting the use of an abstract idea to a particular technological environment. Stating an abstract idea while adding the words ″apply it with a computer″ simply combines those two steps, with the same deficient result. Thus, if a patent's recitation of a computer amounts to a mere instruction to implement an abstract idea on a computer, that addition cannot impart patent eligibility.

The fact that a computer necessarily exists in the physical, rather than purely conceptual, realm, is beside the point. There is no dispute that a computer is a tangible system (in § 101 terms, a ″machine″), or that many computer-implemented claims are formally addressed to patent-eligible subject matter. But if that were the end of the § 101 inquiry, an applicant could claim any principle of the physical or social sciences by reciting a computer system configured to implement the relevant concept. Such a result would make the determination of patent eligibility depend simply on the draftsman's art, thereby eviscerating the rule that laws of nature, natural phenomena, and abstract ideas are not patentable.

2

The representative method claim in this case recites the following steps: (1) ″creating″ shadow records for each counterparty to a transaction; (2) ″obtaining″ start-of-day balances based on the parties' real-world accounts at exchange institutions; (3) ″adjusting″ the shadow records as transactions are entered, allowing only those transactions for which the parties have sufficient resources; and (4) issuing irrevocable end-of-day instructions to the exchange institutions to carry out the permitted transactions. Petitioner principally contends that the claims are patent eligible because these steps ″require a substantial and meaningful role for the computer.″ As stipulated, the claimed method requires the use of a computer to create electronic records, track multiple transactions, and issue simultaneous instructions; in other words, the computer is itself the intermediary.

In light of the foregoing, the relevant question is whether the claims here do more than simply instruct the practitioner to implement the abstract idea of intermediated settlement on a generic computer. They do not.

Taking the claim elements separately, the function performed by the computer at each step of the process is purely conventional. Using a computer to create and maintain ″shadow″ accounts amounts to electronic recordkeeping – one of the most basic functions of a computer. The same is true with respect to the use of a computer to obtain data, adjust account balances, and issue automated instructions; all of these computer functions are well-understood, routine, conventional activities previously known to the industry. In short, each step does no more than require a generic computer to perform generic computer functions.

Considered as an ordered combination, the computer components of petitioner's method add nothing that is not already present when the steps are considered separately. Viewed as a whole, petitioner's method claims simply recite the concept of intermediated settlement as performed by a generic computer. The method claims do not, for example, purport to improve the functioning of the computer itself. Nor do they effect an improvement in any other technology or technical field. Instead, the claims at issue amount to ″nothing significantly more″ than an instruction to apply the abstract idea of intermediated settlement using some unspecified, generic computer. Under our precedents, that is not enough to transform an abstract idea into a patent-eligible invention.

C

Petitioner's claims to a computer system and a computer-readable medium fail for substantially the same reasons. Petitioner conceded below that its media claims rise or fall with its method claims. As to its system claims, petitioner emphasizes that those claims recite ″specific hardware″ configured to perform ″specific computerized functions.″ But what petitioner characterizes as specific hardware – a ″data processing system″ with a ″communications controller″ and ″data storage unit,″ for example – is purely functional and generic. Nearly every computer will include a ″communications controller″ and ″data storage unit″ capable of performing the basic calculation, storage, and transmission functions required by the method claims. As a result, none of the hardware recited by the system claims offers a meaningful limitation beyond generally linking the use of the method to a particular technological environment,[7] that is, implementation via computers.

7. Is a computer itself patentable?

Put another way, the system claims are no different from the method claims in substance. The method claims recite the abstract idea implemented on a generic computer; the system claims recite a handful of generic computer components configured to implement the same idea. This Court has long warned against interpreting § 101 in ways that make patent eligibility depend simply on the draftsman's art. Holding that the system claims are patent eligible would have exactly that result.

Because petitioner's system and media claims add nothing of substance to the underlying abstract idea, we hold that they too are patent ineligible under § 101.

C Copyright

Oracle America, Inc. v. Google Inc.
872 F. Supp. 2d 974 (N.D. Cal. 2012)

This action was the first of the so-called "smartphone war" cases tried to a jury. This order includes the findings of fact and conclusions of law on a central question tried simultaneously to the judge, namely the extent to which, if at all, certain replicated elements of the structure, sequence and organization of the Java application programming interface are protected by copyright.[8]

8. The facts here are drawn primarily from the Court of Appeals' opinion, which immediately follows.

Sun Microsystems, Inc. ("Sun") developed the Java "platform" for computer programming and released it in 1996.[1] The aim was to relieve programmers from the burden of writing different versions of their computer programs for different operating systems or devices. The Java platform, through the use of a virtual machine, enabled software developers to write programs that were able to run on different types of computer hardware without having to rewrite them for each different type. With Java, a software programmer could "write once, run anywhere."

1. Oracle acquired Sun in 2010.

The Java virtual machine ("JVM") plays a central role in the overall Java platform. The Java programming language itself – which includes words, symbols, and other units, together with syntax rules for using them to create instructions – is the language in which a Java programmer writes source code, the version of a program that is in a human-readable language. For the instructions to be executed, they must be converted (or compiled) into binary machine code (object code) consisting of 0s and Is understandable by the particular computing device. In the Java system, source code is first converted into "bytecode," an intermediate form, before it is then converted into binary machine code by the Java virtual machine that has been designed for that device.

Sun wrote a number of ready-to-use Java programs to perform common computer functions and organized those programs into groups it called "packages." These packages, which are the application programming interfaces at issue in this appeal, allow programmers to use the prewritten code to build certain functions into their own programs, rather than write their own code to perform those functions from scratch. They are shortcuts. Sun called the code for a specific operation (function) a "method." It defined "classes" so that each class consists of specified methods plus variables and other elements on which the methods operate. To organize the classes for users, then, it grouped classes (along with certain related "interfaces") into "packages." The parties have not disputed the district court's analogy: Oracle's collection of API packages is like a library, each package is like a bookshelf in the library, each class is like a book on the shelf, and each method is like a how-to chapter in a book.

The original Java Standard Edition Platform ("Java SE") included eight packages of pre-written programs. The district court found, and Oracle concedes to some extent, that three of those packages – java.lang.java.io, and java.util – were "core" packages, meaning that programmers using the Java language had to use them in order to make any worthwhile use of the language. By 2008, the Java platform had more than 6,000 methods making up more than 600 classes grouped into 166 API packages. There are 37 Java API packages at issue in this appeal, three of which are the core packages identified by the district court. These packages contain thousands of individual elements, including classes, subclasses, methods, and interfaces.

Every package consists of two types of source code — what the parties call (1) declaring code; and (2) implementing code. Declaring code is the expression that identifies the prewritten function and is sometimes referred to as the "declaration" or "header." As the district court explained, the "main point is that this header line of code introduces the method body and specifies very precisely the inputs, name and other functionality." The expressions used by the programmer from the declaring code command the computer to execute the associated implementing code, which gives the computer the step-by-step instructions for carrying out the declared function.

For example, one of the Java API packages at issue is "java. lang." Within that package is a class called "math," and within "math" there are several methods, including one that is designed to find the larger of two numbers: "max." To invoke this method from another program (or class), the following call could be included in the program:

```
int a = java.lang.Math.max (2, 3);
```

Upon reaching this statement, the computer would go and find the max method under the Math class in the java.lang package, input '2' and '3' as arguments, and then return a '3,' which would then be set as the value of 'a.' The declaration for the "max" method, as defined for integers, is

```
public static int max (int x, int y) {
```

The word "public" means that other programs can call on it. (If this instead says "private," then it can only be accessed by other methods inside the same class.) The word "static" means that the method can be invoked without creating an instance of the class. (If this instead is an instance method, then it would always be invoked with respect to an object.) The word ″int″ means that an integer is returned by the method. (Other alternatives are ″boolean,″ ″char,″ and ″String″ which respectively mean ″true/false,″ ″single character,″ and ″character string.″) Each of these three parameters is drawn from a short menu of possibilities, each possibility corresponding to a very specific functionality. The word ″max″ is a name and any name (other than a reserved word [i.e, a few names like ″int″ and ″public″ that have specific and rigidly defined roles in the Java language]) could have been used.. The phrase ″(int x, int y)″ identifies the arguments that must be passed into the method, stating that they will be in integer form. The ″x″ and the ″y″ could be ″a″ and ″b″ or ″arg1″ and ″arg2,″ so there is a degree of creativity in naming the arguments. Finally, ″{″ is the beginning marker that tells the compiler that the method body is about to follow. The marker is mandatory.

Although Oracle owns the copyright on Java SE and the API packages, it offers three different licenses to those who want to make use of them. The first is the General Public License, which is free of charge and provides that the licensee can use the packages – both the declaring and implementing code – but must ″contribute back″ its innovations to the public. This arrangement is referred to as an ″open source″ license. The second option is the Specification License, which provides that the licensee can use the declaring code and organization of Oracle's API packages but must write its own implementing code. The third option is the Commercial License, which is for businesses that want to use and customize the full Java code in their commercial products and keep their code secret. Oracle offers the Commercial License in exchange for royalties. To maintain Java's ″write once, run anywhere″ motto, the Specification and Commercial Licenses require that the licensees' programs pass certain tests to ensure compatibility with the Java platform.

The accused product is Android, a software platform that was designed for mobile devices and competes with Java in that market. Google acquired Android, Inc. in 2005 as part of a plan to develop a smartphone platform. Later that same year, Google and Sun began discussing the possibility of Google taking a license to use and to adapt the entire Java platform for mobile devices. They also discussed a possible co-development partnership deal with Sun under which Java technology would become an open-source part of the Android platform, adapted for mobile devices. The parties negotiated for months but were unable to reach an agreement. The point of contention between the parties was Google's refusal to make the implementation of its programs compatible with the Java virtual machine or interoperable with other Java programs. Because Sun/Oracle found that position to be anathema to the ″write once, run anywhere″ philosophy, it did not grant Google a license to use the Java API packages.

When the parties' negotiations reached an impasse, Google decided to use the Java programming language to design its own virtual machine – the Dalvik virtual machine (”Dalvik VM”) – and ”to write its own implementations for the functions in the Java API that were key to mobile devices.” Google developed the Android platform, which grew to include 168 API packages – 37 of which correspond to the Java API packages at issue in this appeal.

With respect to the 37 packages at issue, ”Google believed Java application programmers would want to find the same 37 sets of functionalities in the new Android system callable by the same names as used in Java.” To achieve this result, Google copied the declaring source code from the 37 Java API packages verbatim, inserting that code into parts of its Android software. In doing so, Google copied the elaborately organized taxonomy of all the names of methods, classes, interfaces, and packages – the ”overall system of organized names – covering 37 packages, with over six hundred classes, with over six thousand methods. The parties and district court referred to this taxonomy of expressions as the ”structure, sequence, and organization” or ”SSO” of the 37 packages. It is undisputed, however, that Google wrote its own implementing code [with some exceptions not here relevant].

Google released the Android platform in 2007, and the first Android phones went on sale the following year. Oracle indicated at oral argument that Android phones contain copies of the accused portions of the Android software. Android smartphones rapidly grew in popularity and now comprise a large share of the United States market. Google provides the Android platform free of charge to smartphone manufacturers and receives revenue when customers use particular functions on the Android phone. Although Android uses the Java programming language, it is undisputed that Android is not generally Java compatible. As Oracle explains, ”Google ultimately designed Android to be *incompatible* with the Java platform, so that apps written for one will not work on the other.”

Application of Controlling Law to Controlling Facts

All agree that everyone was and remains free to program in the Java language itself.[9] All agree that Google was free to use the Java language to write its own API. While Google took care to provide fresh line-by-line implementations (the 97 percent), it generally replicated the overall name organization and functionality of 37 packages in the Java API (the three percent). The main issue addressed herein is whether this violated the Copyright Act and more fundamentally whether the replicated elements were copyrightable in the first place.

9. Wait. *Why* is everyone free to program in Java? *Is* everyone free to program in Java?

This leads to the first holding central to this order and it concerns the method level. As long as the specific code written to implement a method is different, anyone is free under the Copyright Act to write his or her own method to carry out exactly the same function or specification of any and all methods used in the Java API. Contrary to Oracle, copyright law does not confer ownership over any and all ways to implement a function or specification, no matter how creative the copyrighted implementation or specification may be. The Act confers ownership only over the specific way in which the author wrote out his version. Others are free to write their own implementation to accomplish the identical function, for, importantly, ideas, concepts and functions cannot be monopolized by copyright.

To return to our example, one method in the Java API carries out the function of comparing two numbers and returning the greater. Google – and everyone else in the world – was and remains free to write its own code to carry out the identical function so long as the implementing code in the method body is different from the copyrighted implementation. This is a simple example, but even if a method resembles higher mathematics, everyone is still free to try their hand at writing a different implementation, meaning that they are free to use the same inputs to derive the same outputs so long as the implementation in between is their own.

Much of Oracle's evidence at trial went to show that the design of methods in an API was a creative endeavor. Of course, that is true. Inventing a new method to deliver a new output can be creative, even inventive, including the choices of inputs needed and outputs returned. But such inventions – at the concept and functionality level – are protectable only under the Patent Act. Under the Copyright Act, no matter how creative or imaginative a Java method 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. To repeat the Second Circuit's phrasing, "there might be a myriad of ways in which a programmer may express the idea embodied in a given subroutine." *Computer Associates Intern., Inc. v. Altai, Inc.* The method specification is the *idea*. The method implementation is the *expression*. No one may monopolize the *idea*.

To carry out any given function, the method specification as set forth in the declaration *must be identical* under the Java rules (save only for the choices of argument names). Any other declaration would carry out some *other* function. The declaration requires precision. Significantly, when there is only one way to write something, the merger doctrine bars anyone from claiming exclusive copyright ownership of that expression. Therefore, there can be no copyright violation in using the identical declarations. Nor can there be any copyright violation due to the *name* given to the method (or to the arguments), for under the law, names and short phrases cannot be copyrighted.

In sum, Google and the public were and remain free to write their own implementations to carry out exactly the same functions of all methods in question, using exactly the same method specifications and names. Therefore, at the method level – the level where the heavy lifting is done – Google has violated no copyright, it being undisputed that Google's implementations are different.

* * *

Even so, the second major copyright question is 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. For example, the Math classes in both systems have a method that returns a cosine, another method that returns the larger of two numbers, and yet another method that returns logarithmic values, and so on. As Oracle notes, the rules of Java did not insist that these methods be grouped together in any particular class. Google could have placed its trigonometric function (or any other function) under a class other than Math class. Oracle is entirely correct that the rules of the Java language did not require that the same grouping pattern (or even that they be grouped at all, for each method could have been placed in a stand-alone class). There was nothing in the rules of the Java language that required that Google replicate the same groupings even if Google was free to replicate the same functionality.

The main answer to this argument is that the overall scheme of file name organization is also a command structure for a system or method of operation of the application programming interface. The commands are (and must be) in the form

```
java.package.Class.method()
```

and each calls into action a pre-assigned function.

That a system or method of operation has thousands of commands arranged in a creative taxonomy does not change its character as a method of operation. Yes, it is creative. Yes, it is original. But it is nevertheless a command structure, a system or method of operation – a long hierarchy of over six thousand commands to carry out pre-assigned functions. For that reason, it cannot receive copyright protection.

* * *

Interoperability sheds further light on the character of the command structure as a system or method of operation. Surely, millions of lines of code had been written in Java before Android arrived. These programs necessarily used the java.package.Class.method() command format. These programs called on all or some of the specific 37 packages at issue and necessarily used the command structure of names at issue. Such code was owned by the developers themselves, not by Oracle. *In order for at least some of this code to run on Android, Google was required to provide 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.

That interoperability is at the heart of the command structure is illustrated by Oracle's preoccupation with what it calls "fragmentation," meaning the problem of having imperfect interoperability among platforms. When this occurs, Java-based applications may not run on the incompatible platforms. For example, Java-based code using the replicated parts of the 37 API packages will run on Android but will not if a

38th package is needed. Such imperfect interoperability leads to a ”fragmentation” – a Balkanization – of platforms, a circumstance which Sun and Oracle have tried to curb via their licensing programs. In this litigation, Oracle has made much of this problem, at times almost leaving the impression that if only Google had replicated all 166 Java API packages, Oracle would not have sued. While fragmentation is a legitimate business consideration, it begs the question whether or not a license was required in the first place to replicate some or all of the command structure. (This is especially so inasmuch as Android has not carried the Java trademark, and Google has not held out Android as fully compatible.) The immediate point is this: fragmentation, imperfect interoperability, and Oracle's angst over it illustrate the character of the command structure as a functional system or method of operation.

Oracle America, Inc. v. Google Inc.

750 F.3d 1339 (Fed. Cir. 2014)

We are mindful that the application of copyright law in the computer context is often a difficult task. On this record, however, we find that the district court failed to distinguish between the threshold question of what is copyrightable – which presents a low bar – and the scope of conduct that constitutes infringing activity.[10] The court also erred by importing fair use principles, including interoperability concerns, into its copyrightability analysis. For the reasons that follow, we conclude that the declaring code and the structure, sequence, and organization of the 37 Java API packages are entitled to copyright protection.

10. Why the Federal Circuit? Because the case originally included patent claims, giving the Federal Circuit appellate jurisdiction. Note that the Federal Circuit here is technically applying Ninth Circuit law.

1. Declaring Source Code

Under the merger doctrine, a court will not protect a copyrighted work from infringement if the idea contained therein can be expressed in only one way. For computer programs, this means that when specific parts of the code, even though previously copyrighted, are the only and essential means of accomplishing a given task, their later use by another will not amount to infringement.

In[11] for example, Nintendo designed a program – the 10NES – to prevent its video game system from accepting unauthorized game cartridges. Nintendo ”chose arbitrary programming instructions and arranged them in a unique sequence to create a purely arbitrary data stream” which ”serves as the key to unlock the NES.” Because Nintendo produced expert testimony ”showing a multitude of different ways to generate a data stream which unlocks the NES console,” we concluded that Nintendo's specific choice of code did not merge with the process.

11. Atari Games Corp. v. Nintendo of Am. Inc., 975 F.2d 832 (Fed. Cir. 1992).

The evidence showed that Oracle had unlimited options as to the selection and arrangement of the 7000 lines Google copied. Using the district court's ”java.lang. Math.max” example, Oracle explains that the developers could have called it any number of things, including ”Math.maximum” or ”Arith.larger.” This was not a situation where Oracle was selecting among preordained names and phrases to create its packages. As the district court recognized, moreover, "the Android method and class names could have been different from the names of their counterparts in Java and still have worked." Because alternative expressions were available, there is no merger.

We further find that the district court erred in focusing its merger analysis on the options available to Google at the time of copying. It is well-established that copyrightability and the scope of protectable activity are to be evaluated at the time of creation, not at the time of infringement. The focus is, therefore, on the options that were available to Sun/Oracle at the time it created the API packages. Of course, once Sun/Oracle created "java.lang.Math.max," programmers who want to use that particular package have to call it by that name. But nothing prevented Google from writing its own declaring code, along with its own implementing code, to achieve the same result. In such circumstances, the chosen expression simply does not merge with the idea being expressed.[7]

7. The district court did not find merger with respect to the structure, sequence, and organization of Oracle's Java API packages. Nor could it, given the court's recognition that there were myriad ways in which the API packages could have been organized.

The district court is correct that words and short phrases such as names, titles, and slogans are not subject to copyright protection. The court failed to recognize, however, that the relevant question for copyrightability purposes is not whether the work at issue contains short phrases – as literary works often do – but, rather, whether those phrases are creative. And, by dissecting the individual lines of declaring code at issue into short phrases, the district court further failed to recognize that an original combination of elements can be copyrightable.

By analogy, the opening of Charles Dickens' *A Tale of Two Cities* is [12] [13] nothing but a string of short phrases. Yet no one could contend that this portion of Dickens' work is unworthy of copyright protection because it can be broken into those shorter constituent components. The question is not whether a short phrase or series of short phrases can be extracted from the work, but whether the manner in which they are used or strung together exhibits creativity.

12. "It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way – in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only." Charles Dickens, *A Tale of Two Cities*

13. "*Mr. Burns*: This is a thousand monkeys working at a thousand typewriters. Soon, they'll have written the greatest novel known to man. Let's see ... 'It was the best of times, it was the blurst of times!' You stupid monkey!" *The Simpsons*, "Last Exit to Springfield" (episode 9F15).

Although the district court apparently focused on individual lines of code, Oracle is not seeking copyright protection for a specific short phrase or word. Instead, the portion of declaring code at issue is 7,000 lines, and Google's own "Java guru" conceded that there can be "creativity and artistry even in a single method declaration." Because Oracle exercised creativity in the selection and arrangement of the method declarations when it created the API packages and wrote the relevant declaring code, they contain protectable expression that is entitled to copyright protection.

2. *The Structure, Sequence, and Organization of the API Packages*

The district court found that the SSO of the Java API packages is creative and original, but nevertheless held that it is a system or method of operation and, therefore, cannot be copyrighted. In reaching this conclusion, the district court seems to have relied upon language contained in a First Circuit decision,[14]

14. Lotus Dev. Corp. v. Borland Int'l, Inc., 49 F.3d 807 (1st Cir. 1995).

In *Lotus Development Corp. v. Borland International, Inc.*, it was undisputed that the defendant copied the menu command hierarchy and interface from Lotus 1-2-3, a computer spreadsheet program that enables users to perform accounting functions electronically on a computer. The menu command hierarchy referred to a series of commands – such as "Copy," "Print," and "Quit" – which were arranged into more than 50 menus and submenus. Although the defendant did not copy any Lotus source code, it copied the menu command hierarchy into its rival program. The question before the court was whether a computer menu command hierarchy is copyrightable subject matter.

Although it accepted the district court's finding that Lotus developers made some expressive choices in selecting and arranging the command terms, the First Circuit found that the command hierarchy was not copyrightable because, among other things, it was a "method of operation" under Section 102(b). In reaching this conclusion, the court defined a "method of operation" as "the means by which a person operates something, whether it be a car, a food processor, or a computer." Because the Lotus menu command hierarchy provided "the means by which users control and operate Lotus 1-2-3," it was deemed unprotectable. For example, if users wanted to copy material, they would use the "Copy" command and the command terms would tell the computer what to do. According to the Lotus court, the "fact that Lotus developers could have designed the Lotus menu command hierarchy differently is immaterial to the question of whether it is a 'method of operation.' The court further indicated that, "[i]f specific words are essential to operating something, then they are part of a 'method of operation' and, as such, are unprotectable."

On appeal, Oracle argues that the district court's reliance on *Lotus Development* is misplaced because it is distinguishable on its facts and is inconsistent with Ninth Circuit law. We agree. First, while the defendant in Lotus did not copy any of the underlying code, Google concedes that it copied portions of Oracle's declaring source code verbatim. Second, the *Lotus Development* court found that the commands at issue there (copy, print, etc.) were not creative, but it is undisputed here that the declaring code and the structure and organization of the API packages are both creative and original. Finally, while the court in *Lotus Development* found the commands at issue were "essential to operating" the system, it is undisputed that – other than perhaps as to the three core packages – Google did not need to copy the structure, sequence, and organization of the Java API packages to write programs in the Java language.

More importantly, however, the Ninth Circuit has not adopted the court's "method of operation" reasoning in *Lotus Development*, and we conclude that it is inconsistent with binding precedent. Specifically, we find that *Lotus Development* is inconsistent with Ninth Circuit case law recognizing that the structure, sequence, and organization of a computer program is eligible for copyright protection where it qualifies as an expression of an idea, rather than the idea itself. We find, moreover, that the hard and fast rule set down in *Lotus Development* and employed by the district court here – i.e., that elements which perform a function can never be copyrightable – is at odds with the Ninth Circuit's endorsement of the abstraction-filtration-comparison analysis.

Here, the district court recognized that the SSO "resembles a taxonomy," but found that "it is nevertheless a command structure, a system or method of operation — a long hierarchy of over six thousand commands to carry out pre-assigned functions."[12] In other words, the court concluded that, although the SSO is expressive, it is not copyrightable because it is also functional. The problem with the district court's approach is that computer programs are by definition functional – they are all designed to accomplish some task. If we were to accept the district court's suggestion that a computer program is uncopyrightable simply because it "carr[ies] out pre-assigned functions," no computer program is protectable. That result contradicts Congress's express intent to provide copyright protection to computer programs, as well as binding Ninth Circuit case law finding computer programs copyrightable, despite their utilitarian or functional purpose.

12. This analogy by the district court is meaningful because taxonomies, in varying forms, have generally been deemed copyrightable. *See, e.g., Practice Management. Info. Corp. v. Am. Med. Ass'n*

On appeal, Oracle does not — and concedes that it cannot — claim copyright in the idea of organizing functions of a computer program or in the "package-class-method" organizational structure in the abstract. Instead, Oracle claims copyright protection only in its particular way of naming and organizing each of the 37 Java API packages. Oracle recognizes, for example, that it "cannot copyright the idea of programs that open an internet connection," but "it can copyright the precise strings of code used to do so, at least so long as other language is available to achieve the same function." Thus, Oracle concedes that Google and others could employ the Java language – much like anyone could employ the English language to write a paragraph without violating the copyrights of other English language writers. And, that Google may employ the "package-class-method" structure much like authors can employ the same rules of grammar chosen by other authors without fear of infringement. What Oracle contends is that, beyond that point, Google, like any author, is not permitted to employ the precise phrasing or precise structure chosen by Oracle to flesh out the substance of its packages — the details and arrangement of the prose.

3. *Google's Interoperability Arguments are Irrelevant to Copyrightability*

Oracle also argues that the district court erred in invoking interoperability in its copyrightability analysis. Specifically, Oracle argues that Google's interoperability arguments are only relevant, if at all, to fair use – not to the question of whether the API packages are copyrightable. We agree.

The district court characterized *Sega Enterprises Ltd. v. Accolade, Inc.* and[15] as "close analogies" to this case. According to the court, both decisions "held that interface procedures that were necessary to duplicate in order to achieve interoperability were functional aspects not copy-

15. Sony Comput. Entm't v. Connectix Corp., 203 F.3d 596 (9th Cir. 2000).

rightable under Section 102(b)." The district court's reliance on *Sega Enterprises* and *Sony Computer Entertainment v. Connectix Corp.* in the copyrightability context is misplaced, however. Both cases were focused on fair use, not copyrightability. In *Sega Enterprises*, for example, the only question was whether Accolade's intermediate copying was fair use. The court never addressed the question of whether Sega's software code, which had functional elements, also contained separable creative expression entitled to protection.

This is not a case where Google reverse-engineered Oracle's Java packages to gain access to unprotected functional elements contained therein. Had Google reverse engineered the programming packages to figure out the ideas and functionality of the original, and then created its own structure and its own literal code, Oracle would have no remedy under copyright whatsoever. Instead, Google chose to copy both the declaring code and the overall SSO of the 37 Java API packages at issue.

We disagree with Google's suggestion that*Sega Enterprises* and *Sony Computer Entertainment* created an "interoperability exception" to copyrightability. Because copyrightability is focused on the choices available to the plaintiff at the time the computer program was created, the relevant compatibility inquiry asks whether the plaintiff's choices were dictated by a need to ensure that its program worked with existing third-party programs. Whether a defendant later seeks to make its program interoperable with the plaintiff's program has no bearing on whether the software the plaintiff created had any design limitations dictated by external factors. Stated differently, the focus is on the compatibility needs and programming choices of the party claiming copyright protection – not the choices the defendant made to achieve compatibility with the plaintiff's program. Consistent with this approach, courts have recognized that, once the plaintiff creates a copyrightable work, a defendant's desire "to achieve total compatibility... is a commercial and competitive objective which does not enter into the ... issue of whether particular ideas and expressions have merged." *Apple Computer, Inc. v. Franklin Computer Corp.*

Whether Google's software is "interoperable" in some sense with any aspect of the Java platform (although as Google concedes, certainly not with the JVM) has no bearing on the threshold question of whether Oracle's software is copyrightable. It is the interoperability and other needs of Oracle – not those of Google – that apply in the copyrightability context, and there is no evidence that when Oracle created the Java API packages at issue it did so to meet compatibility requirements of other pre-existing programs.

Google maintains on appeal that its use of the "Java class and method names and declarations was 'the only and essential means' of achieving a degree of interoperability with existing programs written in the Java language." Indeed, given the record evidence that Google designed Android so that it would *not* be compatible with the Java platform, or the JVM specifically, we find Google's interoperability argument confusing. While Google repeatedly cites to the district court's finding that Google had to copy the packages so that an app written in Java could run on Android, it cites to no evidence in the record that any such app exists and points to no Java apps that either pre-dated or post-dated Android that could run on the Android platform. The compatibility Google sought to foster was not with Oracle's Java platform or with the JVM central to that platform. Instead, Google wanted to capitalize on the fact that software developers were already trained and experienced in using the Java API packages at issue. Google's interest was in accelerating its development process by leveraging Java for its existing base of developers. Although this competitive objective might be relevant to the fair use inquiry, we conclude that it is irrelevant to the copyrightability of Oracle's declaring code and organization of the API packages.

Finally, to the extent Google suggests that it was entitled to copy the Java API packages because they had become the effective industry standard, we are unpersuaded. Google cites no authority for its suggestion that copyrighted works lose protection when they become popular, and we have found none. In fact, the Ninth Circuit has rejected the argument that a work that later becomes the industry standard is uncopyrightable. *See*[16] (noting that the district court found plaintiff's medical coding system entitled to copyright protection, and that, although the system had become the industry standard, plaintiff's copyright did not prevent competitors ″from developing comparative or better coding systems and lobbying the federal government and private actors to adopt them. It simply prevents wholesale copying of an existing system.″). Google was free to develop its own API packages and to lobby programmers to adopt them. Instead, it chose to copy Oracle's declaring code and the SSO to capitalize on the preexisting community of programmers who were accustomed to using the Java API packages. That desire has nothing to do with copyrightability. For these reasons, we find that Google's industry standard argument has no bearing on the copyrightability of Oracle's work.

16. Practice Mgmt. Info. Corp. v. Am. Med. Ass'n, 121 F.3d 516 (9th Cir. 1997).

Google LLC v. Oracle America, Inc.
141 S. Ct. 1183 (2021)

[On remand, the jury found that Google's copying was a fair use. The District Court declined to set aside the verdict, but the Federal Circuit reversed, holding that no reasonable jury could find fair use. The Supreme Court then granted certiorari on both the copyrightability and fair use issues.]

Justice BREYER delivered the opinion of the Court. . . .

Generically speaking, computer programs differ from books, films, and many other ″literary works″ in that such programs almost always serve functional purposes. These and other differences have led at least some judges to complain that ″applying copyright law to computer programs is like assembling a jigsaw puzzle whose pieces do not quite fit.″ . . .

The upshot, in our view, is that fair use can play an important role in determining the lawful scope of a computer program copyright, such as the copyright at issue here. It can help to distinguish among technologies. It can distinguish between expressive and functional features of computer code where those features are mixed. It can focus on the legitimate need to provide incentives to produce copyrighted material while examining the extent to which yet further protection creates unrelated or illegitimate harms in other markets or to the development of other products. In a word, it can carry out its basic purpose of providing a context-based check that can help to keep a copyright monopoly within its lawful bounds. . . .

We turn now to the basic legal question before us: Was Google's copying of the Sun Java API, specifically its use of the declaring code and organizational structure for 37 packages of that API, a "fair use." . . .

A. "The Nature of the Copyrighted Work"

The declaring code at issue here resembles other copyrighted works in that it is part of a computer program. Congress has specified that computer programs are subjects of copyright. It differs, however, from many other kinds of copyrightable computer code. It is inextricably bound together with a general system, the division of computing tasks, that no one claims is a proper subject of copyright. It is inextricably bound up with the idea of organizing tasks into what we have called cabinets, drawers, and files, an idea that is also not copyrightable. It is inextricably bound up with the use of specific commands known to programmers, known here as method calls (such as java.lang.Math.max, etc.), that Oracle does not here contest. And it is inextricably bound up with implementing code, which is copyrightable but was not copied.

Moreover, the copied declaring code and the uncopied implementing programs call for, and reflect, different kinds of capabilities. A single implementation may walk a computer through dozens of different steps. To write implementing programs, witnesses told the jury, requires balancing such considerations as how quickly a computer can execute a task or the likely size of the computer's memory. One witness described that creativity as "magic" practiced by an API developer when he or she worries "about things like power management" for devices that "run on a battery." This is the very creativity that was needed to develop the Android software for use not in laptops or desktops but in the very different context of smartphones.

The declaring code (inseparable from the programmer's method calls) embodies a different kind of creativity. Sun Java's creators, for example, tried to find declaring code names that would prove intuitively easy to remember. They wanted to attract programmers who would learn the system, help to develop it further, and prove reluctant to use another. ("Declaring code ... is user facing. It must be designed and organized in a way that is intuitive and understandable to developers so that they can invoke it"). Sun's business strategy originally emphasized the importance of using the API to attract programmers. It sought to make the API ”open” and ”then ... compete on implementations.” The testimony at trial was replete with examples of witnesses drawing this critical line between the user-centered declaratory code and the innovative implementing code.

These features mean that, as part of a user interface, the declaring code differs to some degree from the mine run of computer programs. Like other computer programs, it is functional in nature. But unlike many other programs, its use is inherently bound together with uncopyrightable ideas (general task division and organization) and new creative expression (Android’s implementing code). Unlike many other programs, its value in significant part derives from the value that those who do not hold copyrights, namely, computer programmers, invest of their own time and effort to learn the API’s system. And unlike many other programs, its value lies in its efforts to encourage programmers to learn and to use that system so that they will use (and continue to use) Sun-related implementing programs that Google did not copy.

Although copyrights protect many different kinds of writing, Leval 1116, we have emphasized the need to ”recogni[ze] that some works are closer to the core of [copyright] than others,” In our view, for the reasons just described, the declaring code is, if copyrightable at all, further than are most computer programs (such as the implementing code) from the core of copyright. That fact diminishes the fear, expressed by both the dissent and the Federal Circuit, that application of ”fair use” here would seriously undermine the general copyright protection that Congress provided for computer programs. And it means that this factor, ”the nature of the copyrighted work,” points in the direction of fair use.

B. ”The Purpose and Character of the Use”

. . . Google copied portions of the Sun Java API precisely, and it did so in part for the same reason that Sun created those portions, namely, to enable programmers to call up implementing programs that would accomplish particular tasks. But since virtually any unauthorized use of a copyrighted computer program (say, for teaching or research) would do the same, to stop here would severely limit the scope of fair use in the functional context of computer programs. Rather, in determining whether a use is ”transformative,” we must go further and examine the copying’s more specifically described ”purpose[s]” and ”character.”

Here Google’s use of the Sun Java API seeks to create new products. It seeks to expand the use and usefulness of Android-based smartphones. Its new product offers programmers a highly creative and innovative tool for a smartphone environment. To the extent that Google used parts of the Sun Java API to create a new platform that could be readily used by programmers, its use was consistent with that creative ”progress” that is the basic constitutional objective of copyright itself.

The jury heard that Google limited its use of the Sun Java API to tasks and specific programming demands related to Android. It copied the API (which Sun created for use in desktop and laptop computers) only insofar as needed to include tasks that would be useful in smartphone programs. And it did so only insofar as needed to allow programmers to call upon those tasks without discarding a portion of a familiar programming language and learning a new one. To repeat, Google, through Android, provided a new collection of tasks operating in a distinct and different computing environment. Those tasks were carried out through the use of new implementing code (that Google wrote) designed to operate within that new environment. Some of the amici refer to what Google did as "reimplementation," defined as the "building of a system ... that repurposes the same words and syntaxes" of an existing system —in this case so that programmers who had learned an existing system could put their basic skills to use in a new one.

The record here demonstrates the numerous ways in which reimplementing an interface can further the development of computer programs. The jury heard that shared interfaces are necessary for different programs to speak to each other. ("We have to agree on the APIs so that the application I write to show a movie runs on your device"). It heard that the reimplementation of interfaces is necessary if programmers are to be able to use their acquired skills. ("If the API labels change, then either the software wouldn't continue to work anymore or the developer ... would have to learn a whole new language to be able to use these API labels"). It heard that the reuse of APIs is common in the industry. It heard that Sun itself had used pre-existing interfaces in creating Java. And it heard that Sun executives thought that widespread use of the Java programming language, including use on a smartphone platform, would benefit the company. . . .

These and related facts convince us that the "purpose and character" of Google's copying was transformative—to the point where this factor too weighs in favor of fair use. . . .

C. "The Amount and Substantiality of the Portion Used"

If one considers the declaring code in isolation, the quantitative amount of what Google copied was large. Google copied the declaring code for 37 packages of the Sun Java API, totaling approximately 11,500 lines of code. Those lines of code amount to virtually all the declaring code needed to call up hundreds of different tasks. On the other hand, if one considers the entire set of software material in the Sun Java API, the quantitative amount copied was small. The total set of Sun Java API computer code, including implementing code, amounted to 2.86 million lines, of which the copied 11,500 lines were only 0.4 percent.

The question here is whether those 11,500 lines of code should be viewed in isolation or as one part of the considerably greater whole. . . .

Several features of Google's copying suggest that the better way to look at the numbers is to take into account the several million lines that Google did not copy. For one thing, the Sun Java API is inseparably bound to those task-implementing lines. Its purpose is to call them up. For another, Google copied those lines not because of their creativity, their beauty, or even (in a sense) because of their purpose. It copied them because programmers had already learned to work with the Sun Java API's system, and it would have been difficult, perhaps prohibitively so, to attract programmers to build its Android smartphone system without them. Further, Google's basic purpose was to create a different task-related system for a different computing environment (smartphones) and to create a platform—the Android platform —that would help achieve and popularize that objective. The "substantiality" factor will generally weigh in favor of fair use where, as here, the amount of copying was tethered to a valid, and transformative, purpose.

We do not agree with the Federal Circuit's conclusion that Google could have achieved its Java-compatibility objective by copying only the 170 lines of code that are "necessary to write in the Java language." In our view, that conclusion views Google's legitimate objectives too narrowly. Google's basic objective was not simply to make the Java programming language usable on its Android systems. It was to permit programmers to make use of their knowledge and experience using the Sun Java API when they wrote new programs for smartphones with the Android platform. In principle, Google might have created its own, different system of declaring code. But the jury could have found that its doing so would not have achieved that basic objective. In a sense, the declaring code was the key that it needed to unlock the programmers' creative energies. And it needed those energies to create and to improve its own innovative Android systems.

We consequently believe that this "substantiality" factor weighs in favor of fair use.

D. Market Effects

The fourth statutory factor focuses upon the "effect" of the copying in the "market for or value of the copyrighted work." 17 U.S.C. § 107(4). Consideration of this factor, at least where computer programs are at issue, can prove more complex than at first it may seem. It can require a court to consider the amount of money that the copyright owner might lose. . . . Making a film of an author's book may similarly mean potential or presumed losses to the copyright owner. Those losses normally conflict with copyright's basic objective: providing authors with exclusive rights that will spur creative expression.

But a potential loss of revenue is not the whole story. We here must consider not just the amount but also the source of the loss. As we pointed out in Campbell, a "lethal parody, like a scathing theatre review," may "kil[l] demand for the original." Yet this kind of harm, even if directly translated into foregone dollars, is not "cognizable under the Copyright Act."

Further, we must take into account the public benefits the copying will likely produce. Are those benefits, for example, related to copyright's concern for the creative production of new expression? Are they comparatively important, or unimportant, when compared with dollar amounts likely lost (taking into account as well the nature of the source of the loss)?

We do not say that these questions are always relevant to the application of fair use, not even in the world of computer programs. Nor do we say that these questions are the only questions a court might ask. But we do find them relevant here in helping to determine the likely market effects of Google's reimplementation.

As to the likely amount of loss, the jury could have found that Android did not harm the actual or potential markets for Java SE. And it could have found that Sun itself (now Oracle) would not have been able to enter those markets successfully whether Google did, or did not, copy a part of its API. First, evidence at trial demonstrated that, regardless of Android's smartphone technology, Sun was poorly positioned to succeed in the mobile phone market. The jury heard ample evidence that Java SE's primary market was laptops and desktops. It also heard that Sun's many efforts to move into the mobile phone market had proved unsuccessful. As far back as 2006, prior to Android's release, Sun's executives projected declining revenue for mobile phones because of emerging smartphone technology. When Sun's former CEO was asked directly whether Sun's failure to build a smartphone was attributable to Google's development of Android, he answered that it was not. Given the evidence showing that Sun was beset by business challenges in developing a mobile phone product, the jury was entitled to agree with that assessment.

Second, the jury was repeatedly told that devices using Google's Android platform were different in kind from those that licensed Sun's technology. For instance, witnesses explained that the broader industry distinguished between smartphones and simpler "feature phones." As to the specific devices that used Sun-created software, the jury heard that one of these phones lacked a touchscreen, while another did not have a QWERTY keyboard. For other mobile devices, the evidence showed that simpler products, like the Kindle, used Java software, while more advanced technology, like the Kindle Fire, were built on the Android operating system. This record evidence demonstrates that, rather than just "repurposing [Sun's] code from larger computers to smaller computers," Google's Android platform was part of a distinct (and more advanced) market than Java software.

Looking to these important differences, Google's economic expert told the jury that Android was not a market substitute for Java's software. As he explained, "the two products are on very different devices," and the Android platform, which offers "an entire mobile operating stack," is a "very different typ[e] of produc[t]" than Java SE, which is "just an applications programming framework." Taken together, the evidence showed that Sun's mobile phone business was declining, while the market increasingly demanded a new form of smartphone technology that Sun was never able to offer.

Finally, the jury also heard evidence that Sun foresaw a benefit from the broader use of the Java programming language in a new platform like Android, as it would further expand the network of Java-trained programmers. In other words, the jury could have understood Android and Java SE as operating in two distinct markets. And because there are two markets at issue, programmers learning the Java language to work in one market (smartphones) are then able to bring those talents to the other market (laptops).

Sun presented evidence to the contrary. Indeed, the Federal Circuit held that the "market effects" factor militated against fair use in part because Sun had tried to enter the Android market. But those licensing negotiations concerned much more than 37 packages of declaring code, covering topics like "the implementation of [Java's] code" and "branding and cooperation" between the firms. In any event, the jury's fair use determination means that neither Sun's effort to obtain a license nor Oracle's conflicting evidence can overcome evidence indicating that, at a minimum, it would have been difficult for Sun to enter the smartphone market, even had Google not used portions of the Sun Java API.

On the other hand, Google's copying helped Google make a vast amount of money from its Android platform. And enforcement of the Sun Java API copyright might give Oracle a significant share of these funds. It is important, however, to consider why and how Oracle might have become entitled to this money. When a new interface, like an API or a spreadsheet program, first comes on the market, it may attract new users because of its expressive qualities, such as a better visual screen or because of its superior functionality. As time passes, however, it may be valuable for a different reason, namely, because users, including programmers, are just used to it. They have already learned how to work with it.

The record here is filled with evidence that this factor accounts for Google's desire to use the Sun Java API. This source of Android's profitability has much to do with third parties' (say, programmers') investment in Sun Java programs. It has correspondingly less to do with Sun's investment in creating the Sun Java API. We have no reason to believe that the Copyright Act seeks to protect third parties' investment in learning how to operate a created work.

Finally, given programmers' investment in learning the Sun Java API, to allow enforcement of Oracle's copyright here would risk harm to the public. Given the costs and difficulties of producing alternative APIs with similar appeal to programmers, allowing enforcement here would make of the Sun Java API's declaring code a lock limiting the future creativity of new programs. Oracle alone would hold the key. The result could well prove highly profitable to Oracle (or other firms holding a copyright in computer interfaces). But those profits could well flow from creative improvements, new applications, and new uses developed by users who have learned to work with that interface. To that extent, the lock would interfere with, not further, copyright's basic creativity objectives. . . .

The uncertain nature of Sun's ability to compete in Android's market place, the sources of its lost revenue, and the risk of creativity-related harms to the public, when taken together, convince that this fourth factor—market effects—also weighs in favor of fair use. . . .

We reach the conclusion that in this case, where Google reimplemented a user interface, taking only what was needed to allow users to put their accrued talents to work in a new and transformative program, Google's copying of the Sun Java API was a fair use of that material as a matter of law.

Problems

Tetris Problem

Your client, Thoth Software, would like to create and sell a version of Tetris for the Digix gaming console. What aspects of the game can Thoth imitate without fear of liability? The name? Falling blocks? The shapes of the blocks? Their colors? Lines that disappear when completely filled in? The music? The graphics around the play field?