Oracle v. Google

William Fisher

February 2020

History

- Sun Microsystems develops Java, a software program that enables programmers efficiently to develop applications (“apps”) and then to allow those apps to run on a variety of hardware systems
- 2005: Google rejects license from Sun and develops Android
- Oracle acquires Sun and brings copyright infringement suit against Google
- Jury finds that Google copied copyrightable expression but fails to resolve fair use issue
- District Court grants judgment notwithstanding the verdict, finding that declaring code and structure of Java are both not copyrightable
- 2014: CAFC reverses, remands for retrial on fair use
- SCOTUS denies certiorari
- 2016: Jury finds fair use
- 2018: CAFC reverses, finding no fair use ”as a matter of law”
- 2019: SCOTUS grants certiorari

Structure of Java

- 30,000 “methods”
  - Each contains a small amount of “declaring code” and a large amount of “implementing code”
- 3000 “classes” of methods
- 166 “packages” of classes

Declaring code v. implementing code

```
Int Math.maxPair(x,y) { (declaring code)
   if(x >= y) return x; (implementing
code)
   else return y;
}
```

vs.

```
Int Math.maxPair(x,y) {
   if(y >= x) return y;
   else return x;
}
```

Same declaring code, different implementations.

Source: Frank Xiao

Java 6 standard edition Package List

Based on the package descriptions available at **http://java.sun.com/javase/6/docs/api/**

Essential Core Packages

java.lang	Classes that are fundamental to the design of the Java programming language.
java.util	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, miscellaneous utility classes (a string tokenizer, a random-number generator).
java.io	System input and output through data streams, serialization and the file system.
java.math	Arbitrary-precision integer (BigInteger) and decimal (BigDecimal) arithmetic.
java.text	For handling text, dates, numbers, and messages in a manner independent of natural languages.
java.text.spi	Service provider classes for the classes in the java.text package.

Concurrency Support

java.util.concurrent	Utility classes commonly useful in concurrent programming.
java.util.concurrent.atomic	Toolkit supporting lock-free thread-safe programming on single variables.
java.util.concurrent.locks	Framework for locking and waiting for conditions that is distinct from the built-in synchronization and monitors.

SQL and Transactions

java.sql	API for accessing and processing data stored in a data source (e.g. a relational database).
javax.sql	Provides the API for server side data source access and processing from Java.
javax.sql.rowset	Standard interfaces and base classes for JDBC RowSet implementations.
javax.sql.rowset.serial	Utility classes to allow serializable mappings between SQL types and Java data types.
javax.sql.rowset.spi	Third party vendor support for their implementation of a synchronization provider.

Miscellaneous Utilities

java.util.jar	Classes for reading and writing the JAR (Java ARchive) file format, which is based on the standard ZIP file format with an optional manifest file.
java.util.logging	Classes and interfaces of the JavaTM 2 platform's core logging facilities.
java.util.prefs	Allows applications to store and retrieve user and system preference and configuration data.
java.util.regex	Classes for matching character sequences against patterns specified by regular expressions.
java.util.spi	Service provider classes for the classes in the java.util package.
java.util.zip	Classes for reading and writing the standard ZIP and GZIP file formats.
javax.script	Scripting API, defines Scripting Engines and provides framework for their use.

Security and Cryptography

java.security	Classes and interfaces for the security framework.
java.security.acl	*This package has been superseded by the java.security package.*
java.security.cert	Handling certificates, certificate revocation lists (CRLs), and certification paths.
java.security.interfaces	Interfaces for generating RSA and DSA keys.
java.security.spec	Classes/interfaces for key specifications and algorithm parameter specifications.
javax.crypto	Classes and interfaces for cryptographic operations.
javax.crypto.interfaces	Interfaces for Diffie-Hellman keys as defined in RSA Laboratories' PKCS #3.
javax.crypto.spec	Classes/interfaces for key specifications and algorithm parameter specifications.
javax.security.auth	This package provides a framework for authentication and authorization.
javax.security.auth.callback	For application interaction, to display info (e.g. error and warning messages) or retrieve information (e.g. authentication data auch as usernames, passwords)
javax.security.auth.kerberos	Utility classes related to the Kerberos network authentication protocol.
javax.security.auth.login	A pluggable authentication framework.
javax.security.auth.spi	Interface to be used for implementing pluggable authentication modules.
javax.security.auth.x500	Classes to store X500 Principal and X500 Private Crendentials in a Subject.
javax.security.cert	Provides classes for public key certificates.
javax.security.sasl	Contains class and interfaces for supporting SASL.
javax.xml.crypto	Common classes for XML cryptography.
javax.xml.crypto.dom	DOM-specific classes for the javax.xml.crypto package.
javax.xml.crypto.dsig	Classes for generating and validating XML digital signatures.
javax.xml.crypto.dsig.dom	DOM-specific classes for the javax.xml.crypto.dsig package.
javax.xml.crypto.dsig.keyinfo	Classes for parsing and processing KeyInfo elements and structures.
javax.xml.crypto.dsig.spec	Parameter classes for XML digital signatures.
org.ietf.jgss	Unified API for using security services (e.g. authentication, data integrity, data confidentiality) from various underlying security mechanisms like Kerberos.

I/O Facilities

java.nio	Defines buffers (data containers), and overviews other NIO packages.
java.nio.channels	Defines channels (connections to entities offering I/O operations, e.g. files and sockets) and selectors (for multiplexed, non-blocking I/O operations).
java.nio.channels.spi	Service-provider classes for the java.nio.channels package.
java.nio.charset	Charsets, decoders, and encoders, for translating between bytes and Unicode chars.
java.nio.charset.spi	Service-provider classes for the java.nio.charset package.

UI Accessibility

javax.accessibility	Contract between UI components and assistive technology that provides access to them.

AWT and Swing GUI facilities

java.awt	Classes for creating user interfaces and for painting graphics and images.
java.awt.color	Classes for color spaces.
java.awt.datatransfer	Interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop transfer of info between two entities linked to GUI elements.
java.awt.event	Interfaces and classes for dealing with events fired by AWT components.
java.awt.font	Classes and interface relating to fonts.
java.awt.geom	Java 2D classes for defining/performing 2-D geometry operations on objects.
java.awt.im	Classes and interfaces for the input method framework.
java.awt.im.spi	Interfaces for input methods that can be used with any Java runtime environment.
java.awt.image	Classes for creating and modifying images.
java.awt.image.renderable	Classes and interfaces for producing rendering-independent images.
java.awt.print	Classes and interfaces for a general printing API.
javax.swing	"Lightweight" (all-Java) components that work similarly on all platforms.
javax.swing.border	Classes and interface for drawing specialized borders around a Swing component.
javax.swing.colorchooser	Classes and interfaces used by the JColorChooser component.
javax.swing.event	Events fired by Swing components.
javax.swing.filechooser	Classes and interfaces used by the JFileChooser component.
javax.swing.plaf	Provides Swing with its pluggable look-and-feel capabilities.
javax.swing.plaf.basic	User interface objects for the Basic look and feel.
javax.swing.plaf.metal	User interface objects for the (default) Java look and feel (once codenamed Metal).
javax.swing.plaf.multi	User interface objects that combine two or more look and feels.
javax.swing.plaf.synth	Synth is a skinnable look and feel in which all painting is delegated.
javax.swing.table	Classes and interfaces for dealing with javax.swing.JTable.
javax.swing.text	Classes and interfaces that deal with editable and noneditable text components.
javax.swing.text.html	Class HTMLEditorKit and supporting classes for creating HTML text editors.
javax.swing.text.html.parser	Default HTML parser, along with support classes.
javax.swing.text.rtf	Class (RTFEditorKit) for creating Rich-Text-Format text editors.
javax.swing.tree	Classes and interfaces for dealing with javax.swing.JTree.
javax.swing.undo	Support for undo/redo in applications such as text editors.

Image and Sound I/O

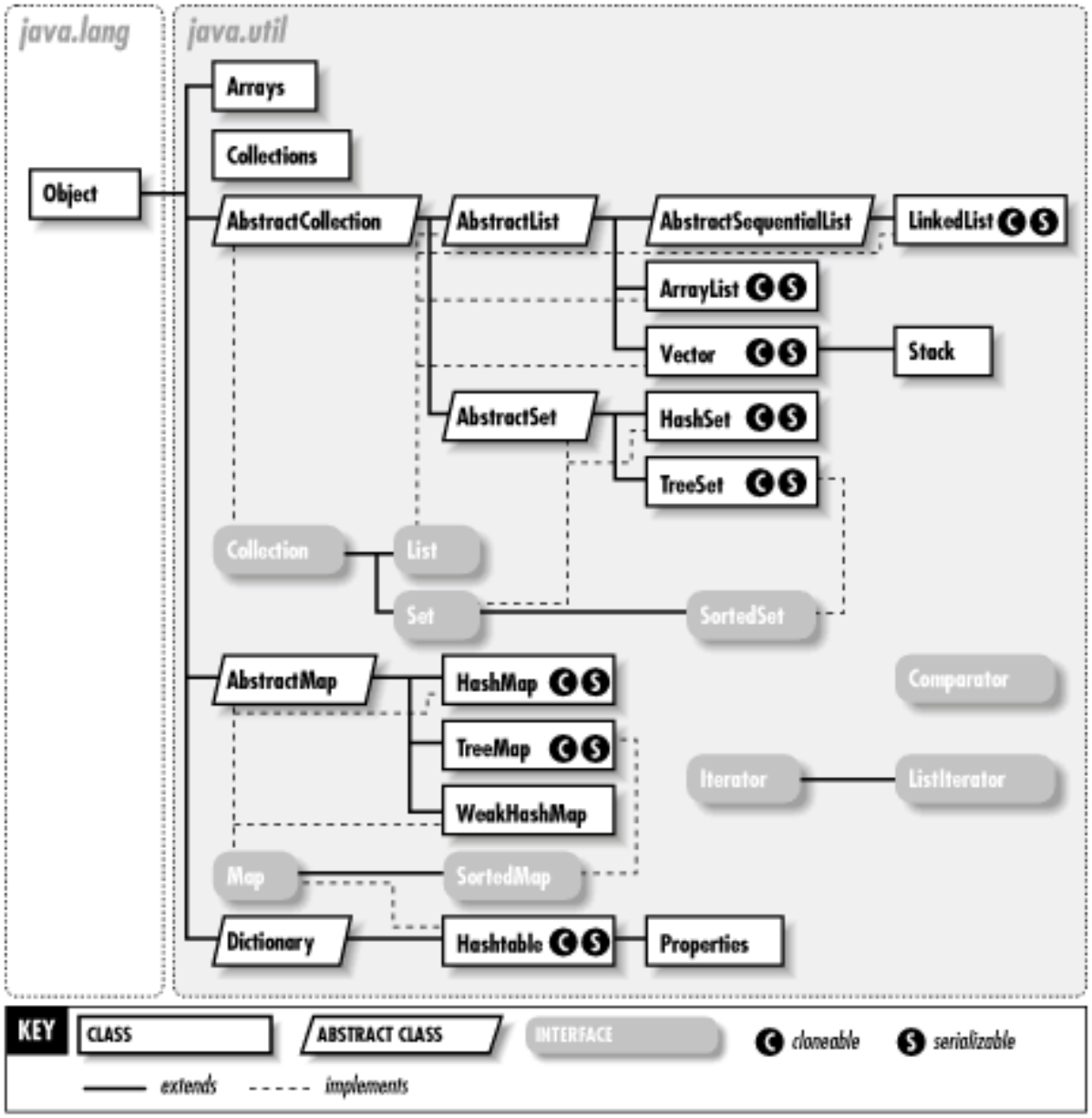
javax.imageio	The main package of the Java Image I/O API.
javax.imageio.event	For synchronous notification of events during the reading and writing of images.
javax.imageio.metadata	Supports reading and writing metadata.
javax.imageio.plugins.bmp	Public classes used by the built-in BMP plug-in.
javax.imageio.plugins.jpeg	Classes supporting the built-in JPEG plug-in.
javax.imageio.spi	Plug-in interfaces for readers, writers, transcoders, streams, & a runtime registry.
javax.imageio.stream	Supports low-level I/O from files and streams.
javax.sound.midi	Interfaces and classes for I/O, sequencing, and synthesis of MIDI data.
javax.sound.midi.spi	Support for new MIDI devices, file readers & writers, sound bank readers.
javax.sound.sampled	Interfaces and classes for capture, processing, and playback of sampled audio data.
javax.sound.sampled.spi	Support for new audio devices, sound file readers & writers, or audio format converters.

Print Service

javax.print	Principal classes and interfaces for the Java Print Service API.
javax.print.attribute	Describing types of Print Service attributes and their collection into attribute sets.
javax.print.attribute.standard	contains classes for specific printing attributes.
javax.print.event	contains event classes and listener interfaces.

Produced by **Dr Peter Dickman**, Dept of Computing Science, **University of Glasgow, UK**. *v6.0 r1 PKG* (2007/06)
See: **http://www.dcs.gla.ac.uk/~pd/JavaRefCard/** Corrections/suggestions/feedback to: **JavaRefCard @ dcs.gla.ac.uk**
NB: The textual descriptions are lightly editted versions of those appearing on Sun's Java website, reproduced *without* permission.

Source: http://www.dcs.gla.ac.uk/~pd/JavaRefCard/private/packagelist.pdf



Source: http://docstore.mik.ua/orelly/java-ent/jnut/ch23_01.htm

Figure 23-1. The collection classes of the java.util package

Source: http://docstore.mik.ua/orelly/java-ent/jnut/ch23_01.htm

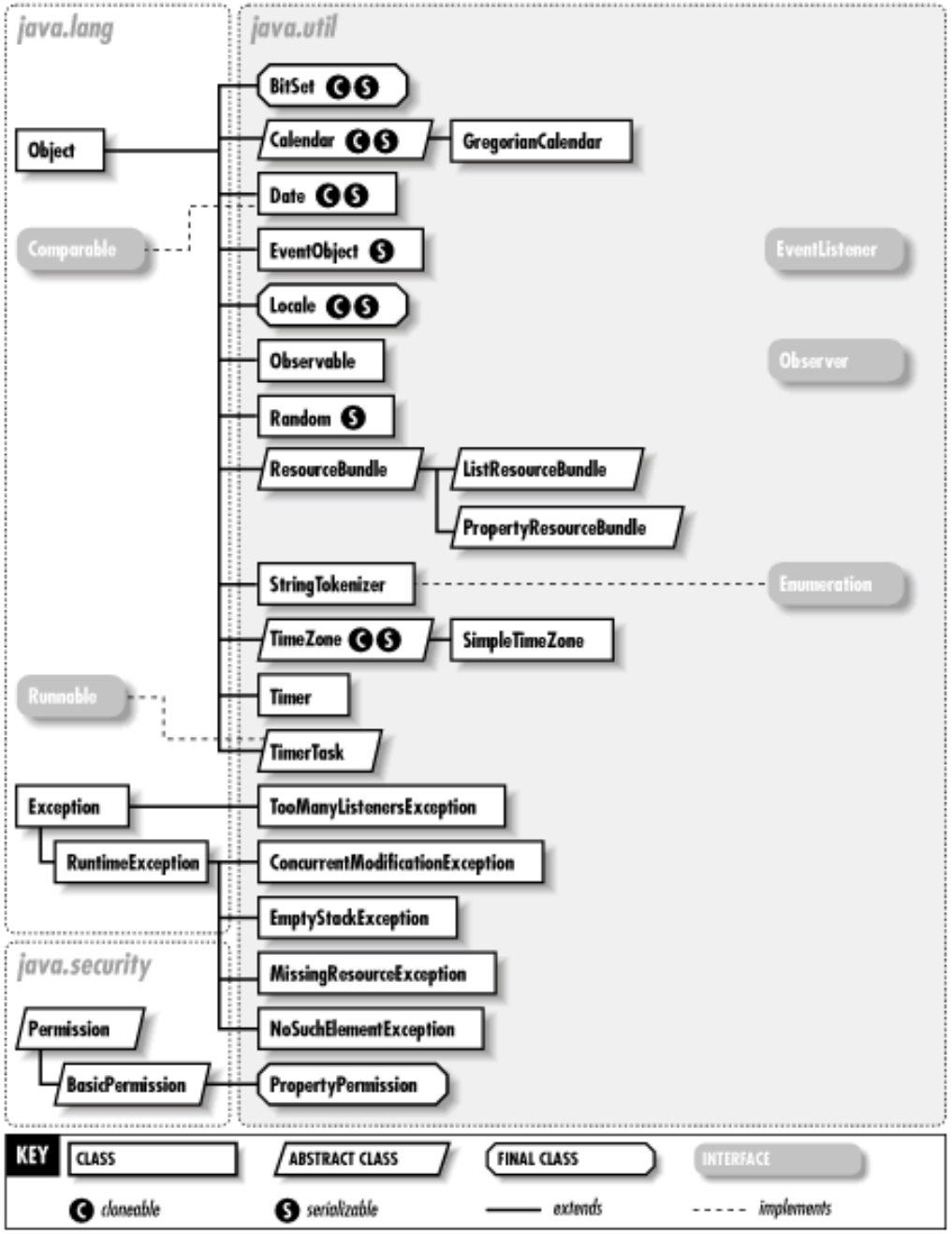


Figure 23-2. Other classes of the java.util package

Sun's Business Model

- Charge hardware manufacturers for the right to install Java on their machines
- Grant royalty-free license to app developers
- Grant royalty-free licenses to developers of competing platforms to use and modify Java so long as they make public all modifications and permit competitors to use those modifications for free
- Charge the developers of competing platforms who wish to keep their modifications proprietary
  - Condition: any modified version of Java must be compatible with Java, so that apps written in Java can run on it
  - Licensees include Blackberry, Nokia, SavaJe, IBM, Oracle

Possible Bases of Liability

Google copied:

1) Structure of 37 (out of 166) Java “Packages” and their components
2) Declaring Code for components of 37 Java packages (7000 lines of source code)
3) Implementing Code for 9 functions within those packages

Altai Test

(1) Abstraction

(2) Filtration

Unprotected Material includes:

(a) Elements dictated by efficiency

(b) Elements dictated by external factors

(i) mechanical specifications of the computer

(ii) compatibility requirements of other programs

(iii) computer manufacturers' design standards

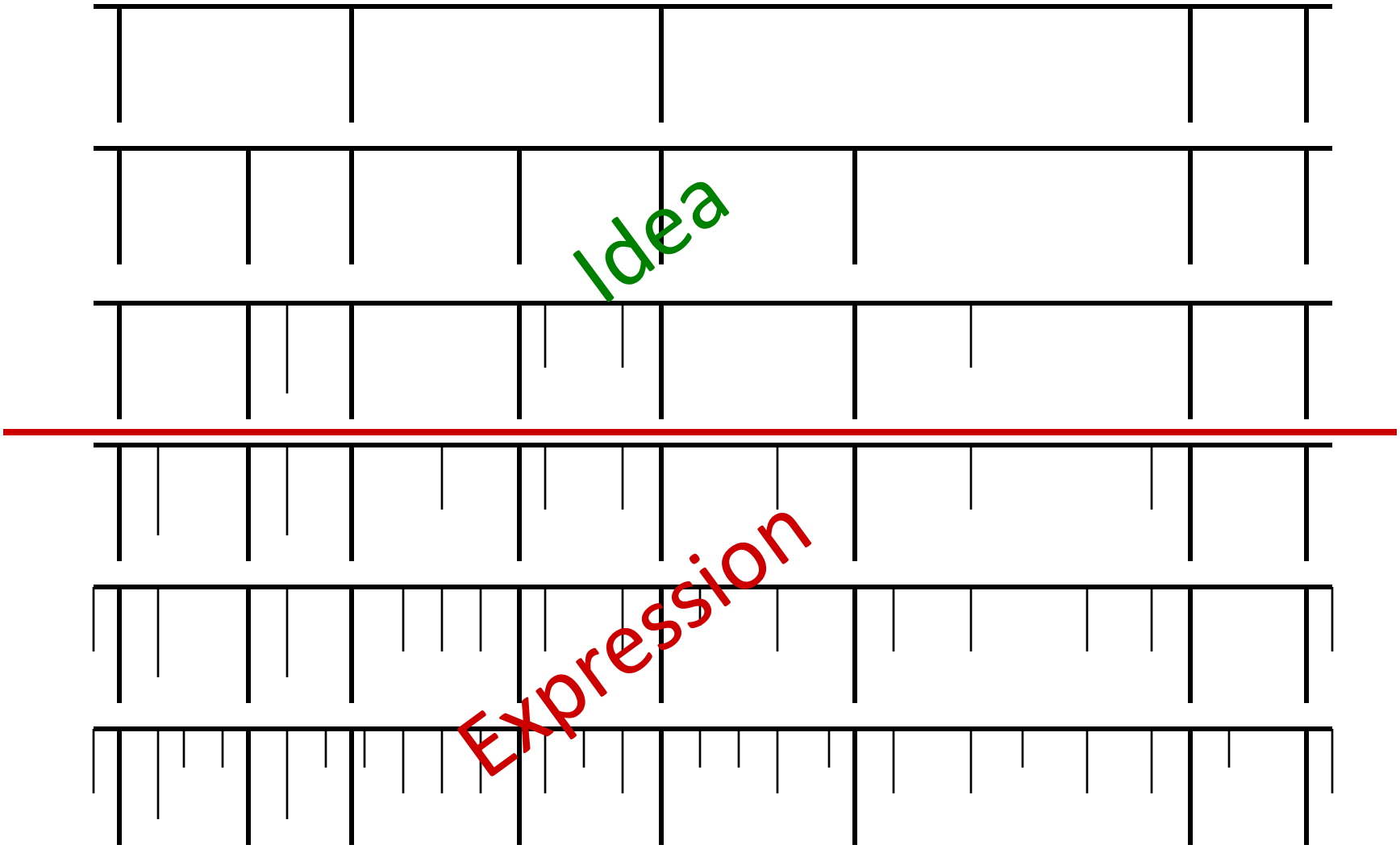
(iv) demands of the industry being served

(v) widely accepted programming practices

(c) Elements taken from public domain

(3) Comparison

Idea & Expression in Plots



Zechariah Chafee (1945)

- "No doubt the line does lie somewhere between the author's idea and the precise form in which he wrote it down. I like to say that the protection covers the "pattern" of the work ... the sequence of events, and the development of the interplay of characters."

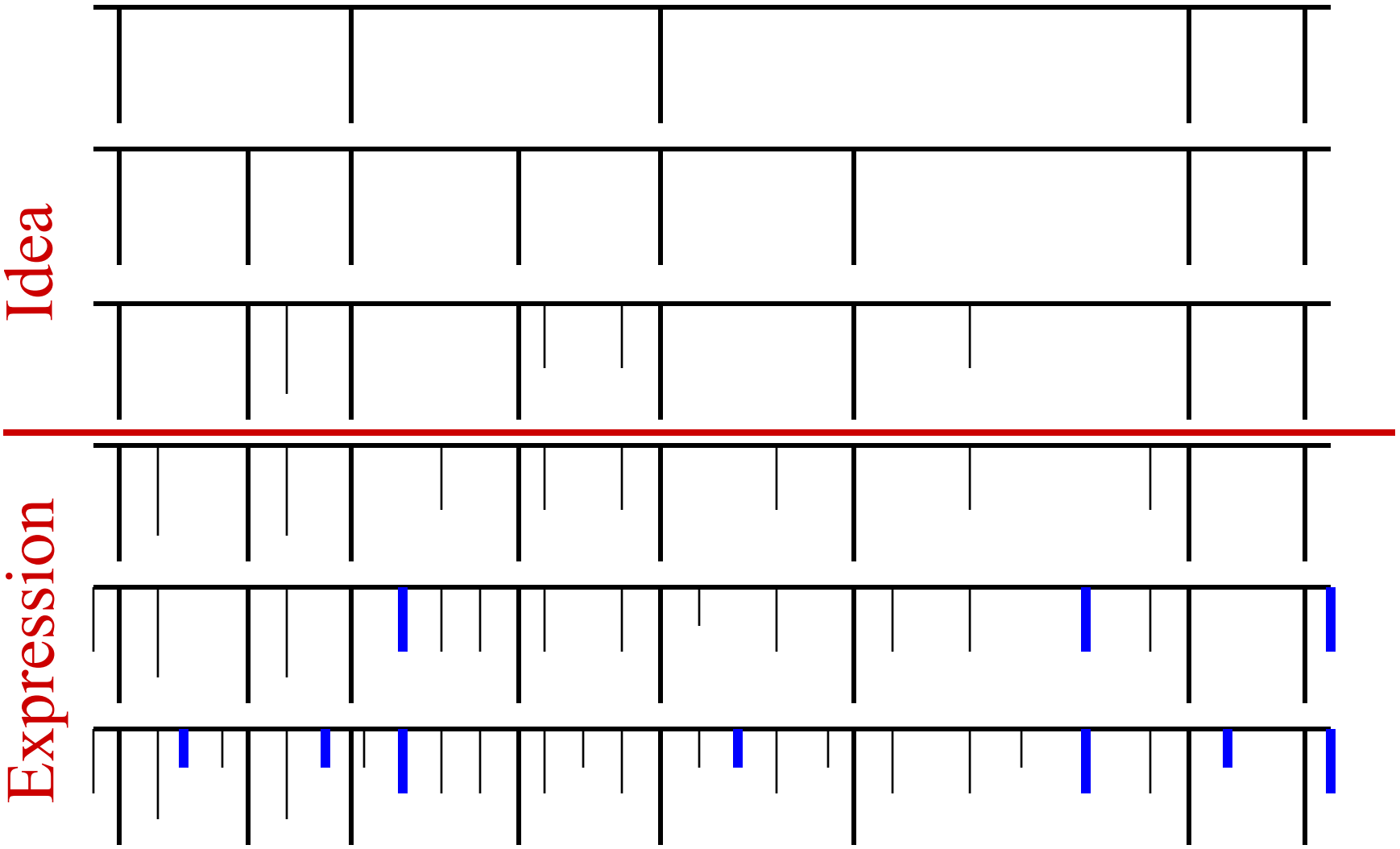
Chiung Yao v. Yu Zheng

Beijing No. 3 Intermediate Court

Dec. 25, 2014

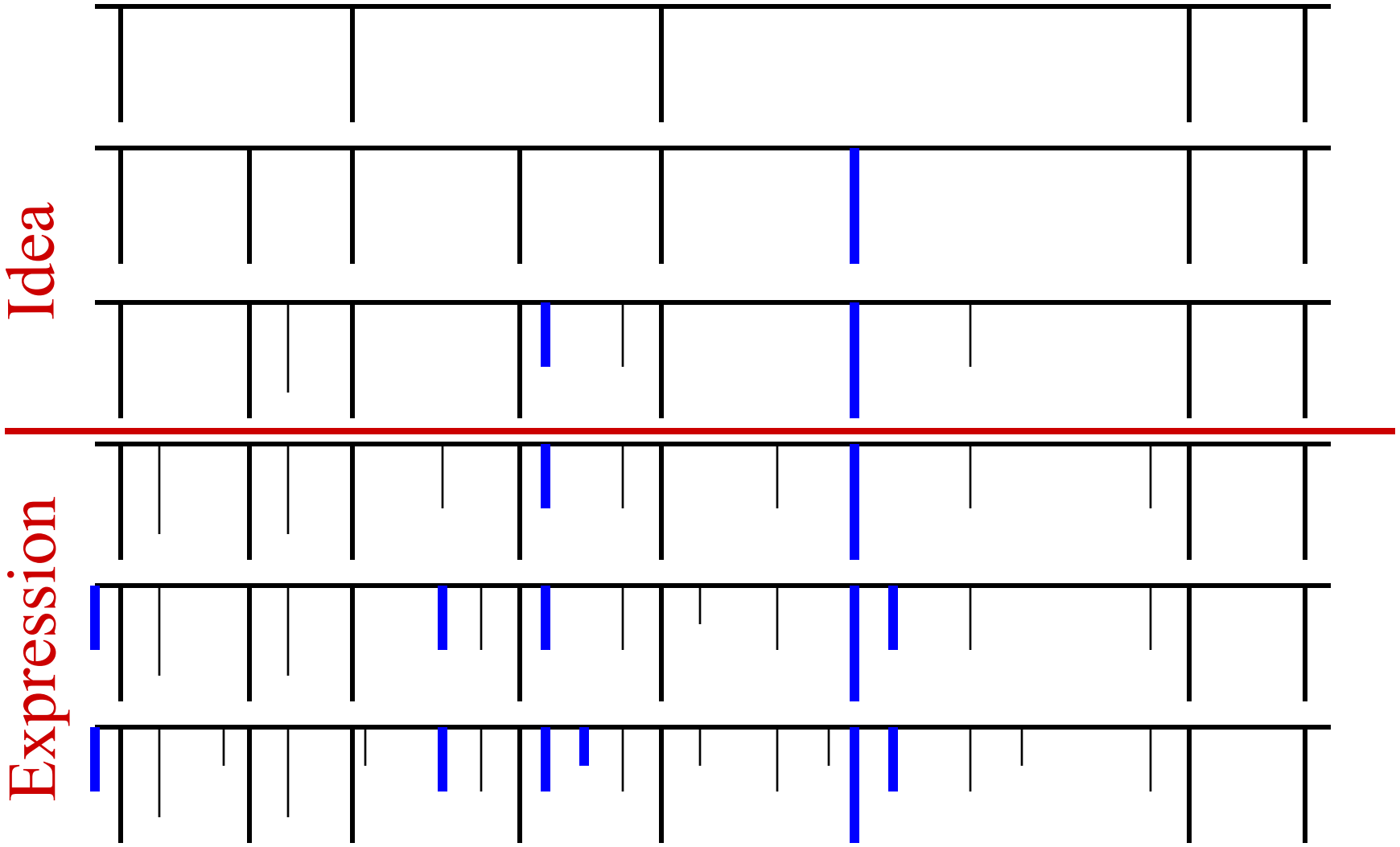
"A plot in a story might be either summarized as an abstract idea, or be presented as a specific expression. Thus, a plot still needs to be further analyzed to determine which part is an idea and what part is expression. To distinguish between an idea and an expression, one shall look at whether the plot in question is abstract and general, or if it is specific enough so that it provides a special aesthetic experience that is sufficient to identify the source of the work. If the plot is specific to such an extent, then it can be considered as an expression."

Filtration

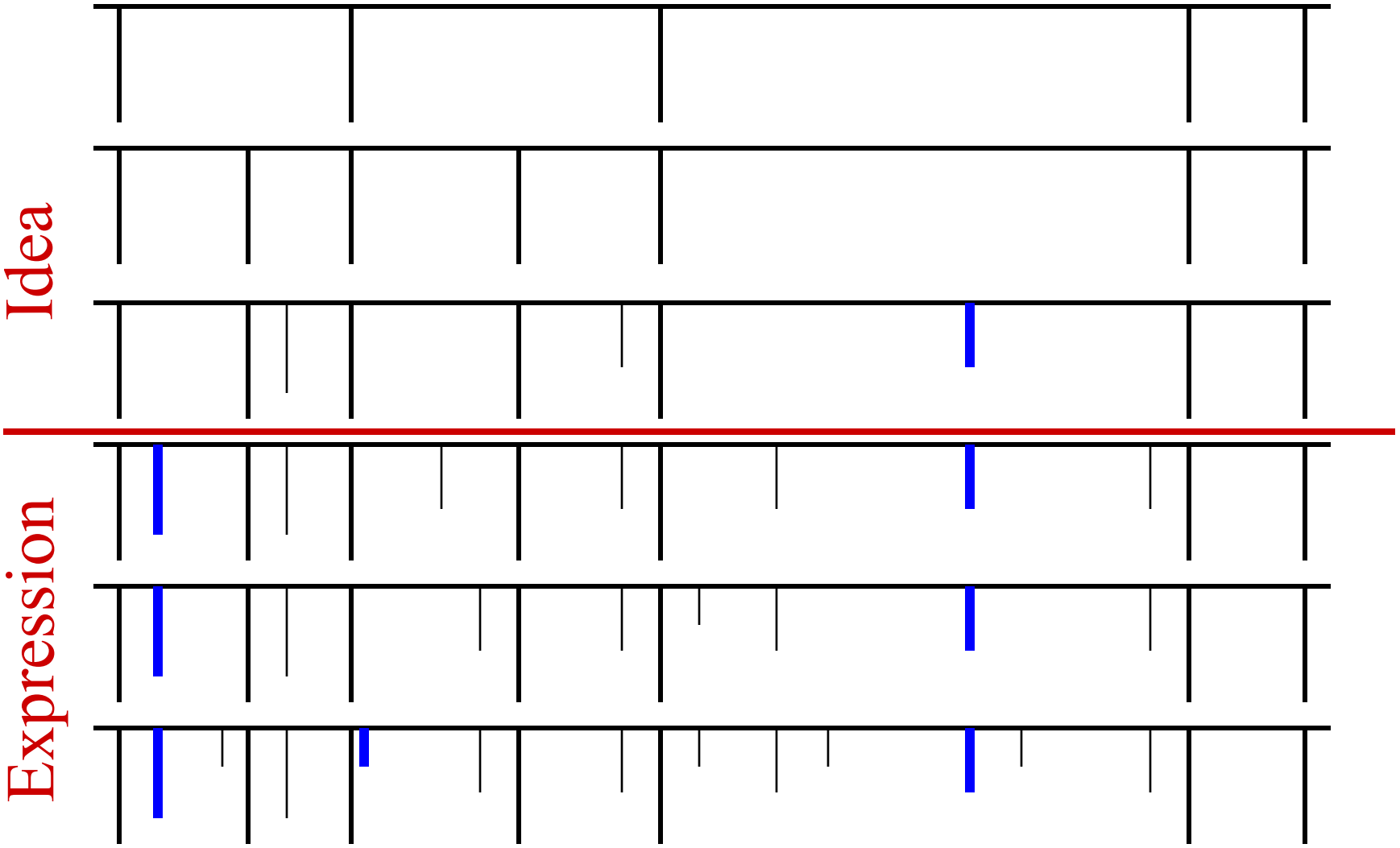


Elements dictated by efficiency

(i) mechanical specifications of the computer
(ii) compatibility requirements of other programs
(iii) computer manufacturers' design standards
(iv) demands of the industry being served
(v) widely accepted programming practices



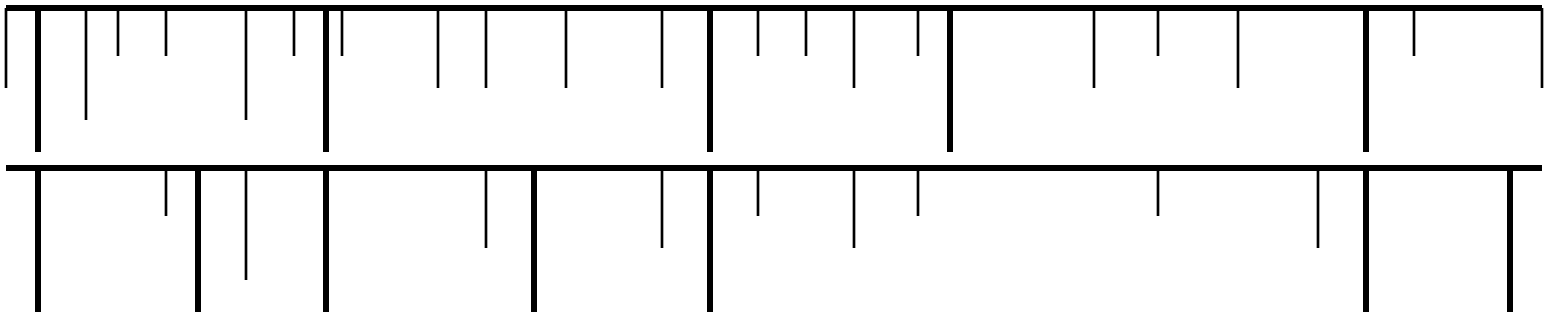
Filtration



Elements taken from public domain

Comparison

Elements of defendant's program



Protected Parts of Plaintiff's program

Possible Arguments for Google

Copyright[X]

1) Lack of Originality
2) Declaring code is not protected "expression"
3) Merger
4) Scene-a-faire
5) No protection for "words and short phrases"
6) Method of Operation – 102(b)
7) De minimis copying
8) Fair Use
9) Privilege for Interoperability
10) Copyright protection for software is bad policy

CAFC (2014), **cert denied 2015**

1) Oracle exercised plenty of creativity
2) Apply *Altai* test to differentiate idea & expression
3) Alternative ways of writing the source code were readily available to G
4) SAF analysis applies at time of creation
5) The arrangement of words is protected
6) **102(b) merely restates the idea/expression distinction**
7) Copying was extensive
8) Remand for new trial on fair use
9) Interoperability may be relevant to fair use
10) Policy is for Congress, not the courts

Possible Arguments for Google

1) Lack of Originality
2) Declaring code is not protected "expression"
3) Merger
4) Scene-a-faire
5) No protection for "words and short phrases"
6) Method of Operation – 102(b)
7) De minimis copying
8) Fair Use
9) Privilege for Interoperability
10) Copyright protection for software is bad policy

Retrial, solely on the issue of fair use
Jury verdict, May 2016: Fair Use
Oracle appeals; oral argument, Dec. 7, 2017
CAFC rejects fair use (March 2018)

8) Remand for new trial on fair use
9) Interoperability may be relevant to fair use

Possible Arguments for Google

1) Lack of Originality
2) Declaring code is not protected "expression"
3) Merger
4) Scene-a-faire
5) No protection for "words and short phrases"
6) Method of Operation – 102(b)
7) De minimis copying
8) Fair Use
9) Privilege for Interoperability
10) Copyright protection for software is bad policy

CAFC (2018)

1) Oracle exercised plenty of creativity
2) Apply *Altai* test to differentiate idea & expression
3) Alternative ways of writing the source code were readily available to G
4) SAF analysis applies at time of creation
5) The arrangement of words is protected
6) **102(b) merely restates the idea/expression distinction**
7) Copying was extensive
8) **No fair use as a matter of law** (applying CA9 law)
9) On remand, Google abandons its reliance on interoperability
10) Policy is for Congress, not the courts

Policy Perspectives

Amicus Brief by Computer Scientists

Copyright[X]

The freedom to reimplement and extend existing APIs has been the key to competition and progress in the computer field—both hardware and software. It made possible the emergence and success of many robust industries we now take for granted—such as industries for mainframes, PCs, peripherals (storage, modems, printers, sound cards, etc.), workstations/servers, and so on—by ensuring that competitors could challenge established players and advance the state of the art.

Thus, excluding APIs from copyright protection has been essential to the development of modern computers and the Internet. For example, the widespread availability of diverse, cheap, and customizable personal computers owes its existence to the lack of copyright on the specification for IBM's Basic Input/Output System (BIOS) for the PC. Companies such as Compaq and Phoenix reimplemented IBM's BIOS without fear of copyright claims, making PC clones possible. And the open nature of APIs was essential to many modern computing developments, including those of operating systems such as UNIX, programming languages such as "C", the Internet's network protocols, and cloud computing.

The uncopyrightable nature of APIs spurs the creation of software that otherwise would not have been written. When programmers can freely reimplement or reverse engineer an API without the need to negotiate a costly license or risk a lawsuit, they can create compatible software that the interface's original creator might never have envisioned or had the resources to create. Moreover, compatible APIs enable people to switch platforms and services freely, and to find software that meets their needs regardless of what browser or operating system they use. Without the compatibility enabled by the open nature of APIs, consumers could be forced to leave their data behind when they switch to a new service.

Amicus Brief by Business Software Alliance

Copyright[x]

[The] growth and innovation [of the software industry in the United States] cannot continue in the absence of clear rules protecting software as copyrighted works. Copyright law must ensure that creativity is rewarded and that intellectual property cannot be misappropriated by those unwilling to pay for the use of another's originality. If developers cannot receive protection for the works they create – or are uncertain about their prospects for protection – then their incentive to create will be removed or reduced, depriving the public and the economy of the benefits of a vibrant software industry.

The decision of the district court in this case threatens to disrupt the well-settled law that has fostered this growth. The particular subject matter here is the copyrightability of Oracle's Java API, but the district court's decision suggests a limited and rigid view of software copyrightability that could have ramifications for all software. The decision thus implicates a major sector of the economy, as well as the benefits we all derive from innovative software at work, at home, and at school.

Microsoft amicus brief

[The District Court's decision denying copyright protection for APIs] sets a dangerous and ill-advised precedent. Under established precedent, sufficiently original software packages like those in the Java platform certainly may be copyrightable, preventing free-riders from replicating their precise structure and suite of features. Yet the District Court's decision leaves no room for that result -- not only in this case but on virtually any facts. To be clear, amici do not suggest that those elements of every computer program are copyrightable, or that copyright in Oracle's Java platform would prevent secondcomers from using the platform to foster further software development or create competing products. Even for copyrightable platforms and software packages, the determination whether infringement has occurred must take into account doctrines like fair use that protect the legitimate interests of follow-on users to innovate. But the promise of some threshold copyright protection for platforms like Java specifically and other elements of computer software generally is a critically important driver of research and investment by companies like amici and rescinding that promise would have sweeping and harmful effects throughout the software industry.

If we gave it away, how can we ensure we get to benefit from it?

Create policies that allow us to drive the standard

- Be the sheppards of the standard we created – we are in the lead because of our head start. Maintaining the pace will guarantee our lead.
- Do not develop in the open. Instead, make source code available after innovation is complete
- Lead device concept: Give early access to the software to partners who build and distribute devices to our specification (ie, Motorola and Verizon). They get a non-contractual time to market advantage and in return they align to our standard.
- We created the first app store for Android and it got critical mass quickly. The store now has value and partners want access to it because of the number of apps available.
- Own the ecosystem we enabled: Evolve the app store. Set the rules. Define developer monetization opportunity. Train developers on our APIs. Give developers one place where they get wide distribution. Provide a global opportunity & payment system. Help developers get distribution via revshare with operators. Extend app store to other devices and other market segments (ie, Google TV)

Takeaway: Provide incentives -- carrots rather than sticks

Peter Menell (2018)

Copyright[X]

"As I explored in my early scholarship, the optimal design of intellectual property protection for addressing the network externality challenge is to protect the functional features of computer software under a limited utility patent-type regime, although with shorter duration and more flexibility to gain access to platforms that become widely adopted. I advocated a genericide-type doctrine which could protect emerging platforms but give way to broader access when a platform becomes dominant and risks affording the proprietor the ability to leverage that control to hinder cumulative innovators. …. At the same time, I opposed copyright protection for the functional and interoperable aspects of computer technology to avoid large returns to first movers that win a standards battle without offering significant technological innovation and to afford competitors to use and build on unpatented methods of operation…. The experience of the past several decades have reinforced the insights of that earlier research. Although copyright law has a valuable role to play in protecting computer software, that role must be limited, especially with regard to network and other functional features of computer software…."

Peter Menell (2018)

Copyright[X]

"We are left with the question of whether the lack of direct *copyright* protection for API design — whose interface must be exposed to the public in most commercial circumstances to be effective— creates an undesirable lacuna in intellectual property protection. Are incentives to innovate platforms inadequate without copyright protection for API design?

"Utility patent law provides protection for novel, non-obvious, and adequately disclosed advances in computer systems, processes, and interface design. It arguably overprotects interface specifications for an excessive duration. Thus, adding robust copyright protection for API design would further undermine realization of network externalities and hamper cumulative innovation….

Thus, looking back over the past three decades, the need for copyright protection to address the dual public goods/network externality problem faced by software developers has substantially waned due to several factors. The emergence and development of the Internet has enabled software developers to distribute software and services at very low cost. Furthermore, developers can protect their code through cloud service models. The Internet has also opened up and expanded the effectiveness of e-commerce and advertising-based business models. More robust copyright protection for API design would likely have stifled platform innovation and competition. Thus, a parsimonious approach to copyright protection of computer software remains the best policy choice. "

Menell's contribution

Copyright[X]

- IP protection for APIs has an undesirable side effect (in addition to the undesirable side effect of all IP rights): forfeiture of the welfare gains associated with network externalities (cf. Boudin in Lotus)
  - However, this undesirable side effect is offset, to some degree, by the tendency of network externalities to retard paradigm shifts in platform technologies
- The need for IP to incentivize innovation with respect to APIs has diminished because of the availability of alternative business models

Implied: present value of the welfare losses associated with slower innovation in platforms is less than the present value of potential welfare gains from network externalities

Problems:
(1) This is just a guess;
(2) The alternative business models have social costs that may exceed the disadvantages of IP

Options

- Copyright Protection; no fair use (Oracle's preference)
- "Genericide" (Menell's early proposal)
  - Modest IP protection initially, but it evaporates once a platform becomes dominant
- Regulatory overlay (cf. management of SEPs)
  - IP owner must disclose copyrights (and patents) on APIs – and announce terms on which they will be licensed
    - IP owner and successors are bound by those commitments
  - FRAND licensing is mandatory (potentially relevant to damages on remand)
- Copyright Protection; fair use, determined on a case-by-case basis (rejected by CAFC)
- No copyright protection (Google's preference)