SOFTIC 2022-1
ソフトウェア等の権利保護に関する調査研究

ソフトウェア等の権利保護に関する調査研究

報告書

— 2022（令和 4）年度 —

2023（令和 5）年 3 月

一般財団法人ソフトウェア情報センター

本事業は、一般社団法人授業目的公衆送信補償金等管理協会（SARTRAS）の共通目的基金の助成を受け実施されています。

序

一般財団法人ソフトウェア情報センター（以下「SOFTIC」という。）は、1986（昭和 61）年の設立から 2007（平成 19）年度までの約 20 年の間、「ソフトウェア等の権利保護に関する調査研究委員会」（委員長：三木茂弁護士）を設置し、主にコンピュータソフトウェア及びその関連領域における内外の裁判例を題材とした調査研究を行ってきた。同委員会は当初、コンピュータソフトウェアの法的保護が著作権法を中心として行われることが確立していった 1980 年代前後の米国裁判例を中心に調査研究を行い、その後、コンピュータソフトウェアそのものに関する裁判例のみならず、1990 年代中頃から急速に一般への普及を見せたインターネットに関連する著作権等に係る諸問題も視点に加え、コンピュータソフトウェアを含むデジタル財とデジタルネットワークに関連する様々な法的問題について先進的な調査研究を行ってきた。

「ソフトウェア等の権利保護に関する調査研究委員会」は、かかる実績を有しつつも、SOFTIC の調査研究事業縮小の結果、2007 年度をもってその歴史に幕を閉じることとなった。

その後 SOFTIC では、各方面の協力により調査研究事業を徐々に再開し、2022 年度までに「ソフトウェア関連発明の特許保護に関する調査研究委員会」において AI を利用した発明の特許化や侵害対応等に関する調査研究を、「OSS 委員会」において OSS の法的諸問題に関する調査研究を、「システム開発紛争判例研究会」において情報システムの開発・導入を巡る紛争事例の調査研究をそれぞれ実施するに至っているが、上記委員会等はそれぞれ分野や内容を特化して調査研究を実施するものであり、「ソフトウェア等の法的保護」にかかる包括的、俯瞰的な観点から、法制度やその運用（裁判例等）などに関して国際的な視点を含めた調査研究を継続的に行う場が存在しない状況にあった。

幸いにも今般、一般社団法人授業目的公衆送信補償金等管理協会（SARTRAS）の助成を受けて、コンピュータソフトウェア及びその関連領域における裁判例等について包括的な検討を行う場として、「ソフトウェア等の権利保護に関する調査研究委員会」を再開することができた。

本委員会では、今後も継続して、SOFTIC の設立以来調査研究事業に関与してきた専門家等の知見の蓄積と比較的若手の専門家等の新たな知見との融合を図ることにより、欧米の動向も含め、従来以上に深い研究成果を世に問うことを目指したいと考えている。

今年度は、2022 年 10 月 31 日の第 1 回から 2023 年 3 月 23 日の第 6 回までと、短期間のうちに多くの委員会を開催することとなった。委員長として委員会の進行を牽引頂いた山神清和教授、ゲストとしてコンピュータソフトウェアの技術的側面についてわかりやすい御説明、御助言をいただいた野山孝太郎様をはじめ、御多忙中のところ御参加いただいた委員、オブザーバ等、関係各位に感謝を申し上げる。

2023（令和 5）年 3 月

一般財団法人ソフトウェア情報センター
専務理事　亀　井　正　博

ソフトウェア等の権利保護に関する調査研究委員会

（2023 年 3 月現在）

委員長

山神　清和	東京都立大学 大学院法学政治学研究科教授

委　員（五十音順）

麻生　典	九州大学 芸術工学研究院准教授
石新　智規	弁護士（シドリーオースティン法律事務所・外国法共同事業）
伊藤　雅浩	弁護士（シティライツ法律事務所）
今村　哲也	明治大学 情報コミュニケーション学部専任教授
岩原　将文	弁護士（岩原法律事務所）
上沼　紫野	弁護士（虎ノ門南法律事務所）
上野　達弘	早稲田大学 法学部教授
大谷　和子	株式会社日本総合研究所 執行役員・法務部長
奥邨　弘司	慶應義塾大学 大学院法務研究科教授
片山　史英	弁護士（虎ノ門南法律事務所）
椙山　敬士	弁護士（虎ノ門南法律事務所）
曽我部　高志	弁護士（水谷法律特許事務所）
谷川　和幸	関西学院大学 法学部准教授
平嶋　竜太	南山大学 法学部・法律学科／法務研究科教授
平野　高志	弁護士（ブレークモア法律事務所）
松尾　剛行	弁護士（桃尾・松尾・難波法律事務所）
宮下　佳之	弁護士（西村あさひ法律事務所）
村尾　治亮	弁護士（東啓綜合法律事務所）

顧問（五十音順）

小川　憲久	弁護士（紀尾井坂テーミス綜合法律事務所
三木　茂	弁護士（スクワイヤ外国法共同事業法律事務所）
水谷　直樹	弁護士・弁理士（水谷法律特許事務所）
吉田　正夫	弁護士（スクワイヤ外国法共同事業法律事務所）

オブザーバ（五十音順）

浅羽　鉄平	富士通株式会社 インフラストラクチャシステム事業本部 Linux ソフトウェア事業部 アプライアンス技術部部長
大内　佳子	富士通株式会社 ビジネス法務・知財本部 知財フロントサービス統括部 知的財産イノベーションセンター
小堀　恵	富士通株式会社 ビジネス法務・知財本部　知財フロントサービス統括部 知的財産イノベーションセンター
森下　元文	文化庁 著作権課課長補佐

事務局

亀井　正博	一般財団法人ソフトウェア情報センター調査研究部長
高橋　宗利	一般財団法人ソフトウェア情報センター調査研究課長
内田　礼	一般財団法人ソフトウェア情報センター調査研究課長代理
中嶋　詩子	一般財団法人ソフトウェア情報センター調査研究部

ソフトウェア等の権利保護に関する調査研究委員会

委員会開催日程及び議事

開催回	開催日	議　事
第 1 回	2022 年 10 月 31 日	1　開会 2　御挨拶 3　委員等紹介 4　委員長選任 5　議事 (1)　趣旨説明（事務局） (2)　今後検討すべき論点・議題について（フリーディスカッション） (3)　その他 6　今後のスケジュールについて 7　閉会
第 2 回	12 月 1 日	1　開会 2　委員、オブザーバ、ゲスト講師紹介 3　議事 ○　Google v. Oracle 事件最高裁判決 4　今後のスケジュールについて 5　閉会
第 3 回	2023 年 1 月 16 日	1　開会 2　議事 ○　Google v. Oracle 事件最高裁判決 3　今後のスケジュールについて 4　閉会
第 4 回	1 月 30 日	1　開会 2　議事 ○　Google v. Oracle 事件最高裁判決 3　今後のスケジュールについて 4　閉会
第 5 回	3 月 16 日	1　開会 2　議事 (1)　Google v. Oracle 事件最高裁判決 (2)　報告書案（作成中）について 3　今後のスケジュールについて 4　閉会
第 6 回	3 月 23 日	1　開会 2　議事 (1)　報告書案（作成中）について (2)　来年度の議題等について 3　今後のスケジュールについて 4　閉会

目　次

第１　2022（令和 4）年度「ソフトウェア等の権利保護に関する調査研究」の概要

１　調査研究の目的

一般財団法人ソフトウェア情報センター（略称：SOFTIC。以下「当財団」という。）では、2022 年度、「ソフトウェア等の法的保護」の観点からの法制度やその運用（裁判例等）などに関する国際的な視点を含めた包括的な調査研究を継続的に行う場として「ソフトウェア等の権利保護に関する調査研究委員会」を設置した。当財団では、同調査研究委員会における調査研究活動を通じて、設立以来調査研究事業に関与してきた専門家等の知見の蓄積と比較的若手の専門家等の新たな知見との融合を図りつつ、欧米の動向も含め、従来以上に深い研究成果を世に問うこととしたい。

本調査研究は、これにより、ソフトウェア等に係る著作権等の保護並びにソフトウェア等の著作物の創作の振興及び普及に資することを目的とするものである。

２　調査研究の内容

(1)　研究対象

「ソフトウェア等の権利保護に関する調査研究」における研究対象となる「ソフトウェア」とは、狭義には「コンピュータソフトウェア」を指すが、本調査研究においては、コンピュータソフトウェアに限らずあらゆるものがコンピュータソフトウェアと同様にデジタル化、ソフトウェア化していく時代において、いわばハードウェア以外の全てのものを研究対象とする意思を表す語として「ソフトウェア等」を捉え、デジタルコンテンツなど各種情報財を中心とする幅広い対象物を研究対象として包含する。

(2)　研究内容

ソフトウェア等の権利保護に係る内外の裁判例、著作権法を中心とする内外の法制度について、担当委員から報告を求め、当該報告について委員会内で議論、検討し、その結果を報告書として取りまとめる。

３　調査研究委員会の開催

ソフトウェア等の権利保護に関する調査研究委員会（以下「委員会」という。）は、ソフトウェア等情報財に関する専門的な研究を行う研究者、法律事務等に従事する実務家のほか、著作権法一般についての研究実績を有する研究者を委員として構成された。ほかに、長年にわたりソフトウェア等の権利保護に関する実務に関わり、当財団の研究員等としての活動実績も有する法律実務家を顧問として迎え、また、当財団賛助会員企業からオブザーバの派遣を得ている。

今年度は、2022 年 10 月 31 日に最初の委員会を開催、2023 年 3 月 23 日まで計 6 回の委員会を開催した。

４　調査研究委員会で検討すべき論点等

(1)　論点・議題の案について

初回となる第 1 回委員会（2022 年 10 月 31 日開催）では、当財団専務理事・調査研究部長からの挨拶、委員等参加者の自己紹介が行われた後、委員の互選により山神清和・東京都立大学大学院法学政治学研究科教授を委員長に選任した。委員長の選任後は、委員長により委員会の進行が行われ、最初に事務局から委員会の設置趣旨等の説明が行われた。

次に、委員会で今後検討すべき論点や議題についての検討がフリーディスカッションの形式で行われ、今後検討すべき論点・議題について、各委員から次のような案が示された。

ア　“Java API”の著作物性等が争われた “Google LLC v. Oracle America, Inc,, 141 S. Ct. 1183 (2021)”の議論を参考としたコンピュータプログラムの創作性に関する検討

○　この研究会に名前がついていますが、「ソフトウェア等」の部分、典型的なコンピュータプログラムの著作物性というのは、整理してもあまり面白みがないかもしれないということでございます。むしろそこから漏れている、もしくは境界にある、プログラムかどうかよくわからないと

いう辺りに、もしかしたら面白いところがあるのではないかということで、多分、この手のことを研究されている方は、非常に注目をされているはずの Google 対 Oracle の最高裁判決ですね。いわゆる API について、実は事前に奥邨委員、それに石新委員と御相談して、これは取り上げると面白いのではないかということを、お話をしたところであります。（山神委員長）

○　Google 対 Oracle の事案というのは、実はアメリカでも結構議論があるようで、フェアユース法理一般との関係で Google 対 Oracle は結構ゴリ押しではないかというような批判も結構あるということは聞いています。実は今、本来のフェアユース的な議論が最高裁でされたように聞いています。Google 対 Oracle というのはかなりフェアユースではイレギュラーな形ではないかみたいな議論もあるようなので、Google 対 Oracle のところからソフトウェア個別の、特化したフェアユースの話なのかという話も広げられるということもあると思います。」（平嶋委員）

イ　音楽教室最高裁判決をベースとした侵害主体論の検討

○　最一小判令和 4 年 10 月 24 日令和 3 年（受ケ）第 1112 号（裁判所 Web）［音楽教室上告審］における音楽教室の生徒の演奏についてその演奏主体をどう考えるかの判示部分は、ICT の発展に水を差す可能性のあるカラオケ法理から、どの程度かい離したものなのか、検討を加える。（山神委員長）

ウ　AI 学習のための著作物の利用と法第 30 条の 4 の妥当性

○　30 条の 4 で、いろいろな要件を勘案した上で、この範囲であれば AI の学習をさせてもよい、そのための利用は大丈夫ですよという話になったとは思われるところ、いろいろな最近の動き、例えば NovelAI Diffusion と言われるようなお絵描き AI が、多分イラスト版漫画村と言われるような、そういうサイトの資料を使っているといった話で、「本当に仕方ないのですか？」ということを問われる時代になっていると思います。現行法の解釈論で頑張るとすると、30 条の 4 のただし書、「ただし、当該著作物の種類及び用途並びに当該利用の態様に照らし著作権者の利益を不当に害することとなる場合は，この限りでない」の解釈論で、何らかの合理的な範囲を作るのかなと思っておりますが、そういう解釈を明確化するというのは、1 つのあり得る方向性で、場合によってはこの委員会というのは別に現行法の解釈だけではなくて、立法の提案もできるのかなと思っておりますので、もし現行法の解釈で適切でなければ立法の提案もあり得るかなと思ったというのが 1 つ目でございます。（松尾委員）

エ　ライセンス契約等による権利制限規定のオーバーライド

○　いわゆるライセンス契約等による権利制限規定のオーバーライドについて議論されていると承知しております。こうしたオーバーライド問題に対して、一定の整理は与えられているとは思うものの、その整理に対してどのように議論をしていくのか。そういう対応の部分につきましては、多分、一定程度は既に議論がされているということだと思うものの、その議論というものが、そのまま、これでいいですよという話なのか、ここ 1、2 年の新たな状況を踏まえて、もし更に検討すべき部分があるのであれば、この委員会でも議論をしていくような余地もあるのかなと思ったところです。（松尾委員）

オ　AI 時代のシステム開発と著作権の問題

○　もしこの委員会で AI 時代のシステム開発と著作権の問題について、もしもう少し検討する機会があれば、有り難いなと思っております。これはジャストアイディアで、こういうことをやってくださると私がうれしいですというものに近いので、是非これをしたいというよりは、そういうことを思っている委員もいますというふうに捉えていただければと思います。（松尾委員）

カ　メタバース空間における著作物の利用

○　ユーザーが何か成果物を作れるようなツールを与えられて、それを使って何かを作るということになったとすると、こういう主体性の問題もテーマになるし、いろいろこう、従来の議論の延長線の問題としての考え方の整理みたいなことも最近の判例を紹介しながらできるのかなという気もするので、もし可能であれば、そういう方向性についてもほかの先生方の御意見を伺えればと思います。（宮下委員）

キ　海外サーバーを使った著作物の利用についての考え方

○　ドワンゴ対 FC2 の事件のように海外サーバーを使った著作物の利用についての考え方、FC2 の事件を参照しながら、著作権保護の観点から見直してみようみたいなこととか、そういう新しいテーマについても議論できるといいのかなと思ったりもしました。（宮下委員）

ク　海外におけるエンフォースメント

○　外国にサーバーがあるためにエンフォースメントの壁に突き当たっていると、そういう問題をどういうふうに理論的に解決するのか、あるいは理論的に突き詰められないところ、例えばどういう国際協調のフレームワークが必要なのかといった課題を整理するということも意味があると思っておりまして、メタバースなどの議論のついでにそういうエンフォースメント関係の視点も入れていただくことができれば、有意義なものになるのではないかと思っております。（大谷委員）

ケ　著作物の自動生成と著作権

○　ローコード、ノーコード、実際プログラムを人が自分で書かなくなっている時代になってきているということで、そういったものの扱いなども実務的には非常に興味があって、プログラムらしきものというのをまとめて検討するというところも非常に親和性もあるということと、実務的にも非常に悩ましいことが起きているので、有り難いと私は思っております。

特に最近だと AI が作り出した画像とか音楽については、誰にどういう権利が帰属するのかという議論が割と盛んに行われてきたところがあると思っていますけれども、一方で、AI がプログラムを書くというか、自動生成するというようなところについては題材としては余り挙がっていないということもありますので、その辺りを少し触れてみるというのも面白いかなと思いました。（伊藤委員）

⑵　今年度検討する論点・議題について

以上のような各種の提案等について検討したところ、今年度に関しては、米国の Google LLC v. Oracle America, Inc. 141 S. Ct. 1183 (2021) 事件（以下「Google. v. Oracle 事件」という。）における「Java API」に関する議論を素材として、コンピュータプログラムの著作物性や権利制限規定（米国における「フェアユース」を含む。）の適用等に関する検討を行うこととされた。

第２　コンピュータプログラムにおける「創作性」及び「権利制限」の考え方に関する検討−Google v. Oracle 事件を素材として−

１　はじめに

前述のとおり、今年度の「ソフトウェア等の権利保護に関する調査研究委員会」では、米国における「Google v. Oracle 事件」を素材として、コンピュータプログラムの著作物性や権利制限規定（米国における「フェアユース」を含む。）の適用等に関する検討を行うこととされた。

Google v. Oracle 事件については、コンピュータプログラムに関する数年ぶり、あるいは数十年ぶりの注目事件として、当財団においてもニュースレター「SOFTIC Law News」により 4 回にわたって報じたほか、2021 年 4 月にあった米国連邦最高裁判所判決の翌 5 月には「Google v. Oracle 事件セミナー」と題するセミナーを実施するなど、注視してきた。

上記 SLN や本調査研究委員会における報告でも触れられているとおり、2021 年 4 月の米国連邦最高裁判所判決は、Oracle が著作権を有する「Java」と称するプログラム言語に係る、同裁判所が判決中で「Java API」と呼ぶもののうち、「Declaring Coce」（宣言コード）と呼ばれる部分を Google が Android OS に利用したことが Oracle の著作権を侵害することになるか否かという点を主な争点として争われたが、連邦巡回区控訴裁判所（CAFC）が 2014 年 5 月の判決で「Declaring Code」に関して著作物性を認めたことは、「驚きをもって受け止められ」た[1]。

裁量上訴の申立てを受理した米国連邦最高裁判所は、この著作物性の論点について何らかの判断を示すのではないかと考えられたが、最終的には「Declaring Code」の著作物性について独自の判断を示すことはなく、これが著作物であると仮定した上でフェアユースの適用可能性を検討し、Google による「Declaring Code」の利用はフェアユースに当たると判断、連邦巡回区控訴裁判所（CAFC）に差し戻した。

本委員会では、「Google v. Oracle 事件」に関して米国連邦最高裁判所が示した判断の相当性についてはおくとして、そもそも、米国連邦裁判所が「Java API」というときの「API」とは、一般的にいうところの「API」[2]と同様のものなのであろうか、技術的に正確な理解を前提とした議論がされているのであろうか、という根本に立ち返った検討[3]を行いつつ、そうした検討を踏まえて、「Java API」の著作物性や権利制限規定の適用可能性について、我が国著作権法にも照らしつつ議論を行った。

以下は、委員会における議論の模様を、ほぼそのままの形で掲載するものである[4]。本委員会における生の議論を世に示すことで、各方面における更なる議論の発展に資することを期待したい。

各報告に用いられた資料は、本報告書のレイアウトの都合上、本報告書本体への掲載は割愛している。別途、本報告書の「資料編」に掲載しているので、参照願いたい。

なお、委員会の場では報告されなかったが、2021 年に米国連邦最高裁判所の判決が示された後の経緯について、参考までに簡単に紹介する。

連邦最高裁判所差戻しを受けた連邦巡回区控訴裁判所（CAFC）は、2021 年 5 月 14 日、本件に係る決定（order）を行った。その内容は次のとおり（主文のみ仮訳。）。

(1)　2018 年 9 月 4 日付けで発行された当裁判所の命令（mandate）は、フェアユースに関してのみ、無効とする（recalled）。

---

[1] 石新智規「ORACLE AMERICA, INC v. GOOGLE INC 米連邦控訴審裁判所（CAFC）2014 年 5 月 9 日判決〜アプリケーションプログラミングインターフェースの著作物性が肯定された事例〜」SLN No. 138（SOFTIC、2014 年 8 月）2 頁。

[2] 「API」とは、「Application Program Interface」の頭字語で、一般的には「オペレーティングシステムやアプリケーションソフトが、他のアプリケーションソフトに対し、機能の一部を利用できるよう提供するインターフェース。一般に、ファイル制御、文字制御、メモリー管理など、さまざまなアプリケーションソフトにとって共通で、かつ利用頻度が高い一連の手続きや関数の集まりを提供する。」（デジタル大辞泉）などと説明される。

[3] 検討に当たって、「Java」について専門的な知見を有するソフトウェア技術者（富士通株式会社 ジャパン・グローバルゲートウェイ アドバンスドテクノロジー推進統括部 野山孝太郎氏）をゲスト講師として招聘し、第 2 回委員会では「Java」及び米国連邦裁判所が「Java API」と呼ぶものの実体について、詳細な説明を受けた。また、同氏には第 3 回委員会及び第 4 回委員会も含め計 3 回にわたって参加を得て、貴重な助言を受けた。

[4] 第 2 回委員会から第 5 回委員会までの議事を掲載している。第 1 回委員会及び第 6 回委員会は、それぞれ、主にテーマ選定並びに今年度報告書及び来年度の検討課題について検討した回であることから、議事の掲載はテーマにかかる上記第 1、4(1)に記載の議事以外を省略している。

(2) 控訴 No. 2017-1118 は復活させる（reinstated）。
(3) フェアユースの問題に関する当裁判所の 2018 年 3 月 27 日付け判決は取り消す（vacated）。
(4) Google に有利となる地方裁判所の終局判決（final judgment）は維持する（affirmed）。
(5) （省略）

連邦巡回区控訴裁判所（CAFC）による本決定により、同裁判所による 2018 年 9 月 4 日発行の命令（mandate[5]）に含まれる同裁判所による 2018 年 3 月 27 日付け判決[6]のうち、フェアユースの成立を否定した部分は無効とされる一方で、フェアユースの成立を認めた連邦地方裁判所の判決[7]及び連邦巡回区控訴裁判所（CAFC）による 2018 年 3 月 27 日付け判決のうち declaring code の著作物性を認めた部分は維持されることとなった。

〔SOFTIC 調査研究部〕

[5] 判決主文に係る認証謄本（certified copy）、裁判所の意見（court's opinion）の写し等から構成される。本件の場合、再審理の申立て等が棄却された 2018 年 8 月 28 日から 7 日後となる同年 9 月 4 日に発行（issue）された。米国連邦控訴規則（Federal Rules of Appellate Procedure）第 41 条参照。

[6] *Oracle Am., Inc. v. Google LLC*, 886 F.3d 1179 (Fed. Cir. 2018). SLN No. 159 参照。

[7] *Oracle Am., Inc. v. Google Inc.*, No. 3:10-cv-3561 (N.D. Cal. June 8, 2016). SLN No. 159、1〜2 頁参照。

２　第 2 回委員会−「Java」及び「Java API」の概要／Google v. Oracle 事件米国連邦最高裁判所判決の概要／「Java API」の著作物性に関する米国連邦裁判所下級審の判断

(1) サマリー

第 2 回委員会（2022 年 12 月 1 日開催）では、はじめに奥邨委員から「Google v. Oracle 事件最高裁判決」と題する資料に基づき Google .v Oracle 事件の経緯に関する概要説明が行われ、続いてゲスト講師として招聘した富士通株式会社ジャパン・グローバルゲートウェイ アドバンスドテクノロジー推進統括部シニアディレクター・野山孝太郎氏から、「Java（ジャバ／ジャヴァ）概説」と題する資料に基づき、「Java」と称するプログラミング言語の概要及び Google .v Oracle 事件に係る判決文中「Java API」と呼ばれるものの内容等について、ソフトウェア技術の観点からの説明が行われた。

以上の報告により事案の概要と技術的な背景について認識を共有した後、再度奥邨委員から Google v. Oracle 事件米国連邦最高裁判所の概要について、さらに石新委員から「Google v. Oracle 事件の経過」と題する資料に基づき、「Java API」の著作物性に関する下級審の判断について、それぞれ説明が行われた。

今回の委員会は以上の各報告が行われた時点で予定された時間を経過してしまったことから、事案の概要に係る質疑等は次回第 3 回以降に行うこととされた。

なお、委員会における報告順は上記のとおりであるが、以下では本件判決で議論の対象となっている「Java」と称するプログラミング言語の概要や判決文中「Java API」と呼ばれるもの内容についての報告及びディスカッションを始めに掲載し、その後、本件判決の概要等の報告を掲載することとしたい。

(2) ディスカッション

ア　「Java」及び「Java API」に関する技術解説

【山神】　判決文中で「Java API」 といわれているものがいったい何なのかということについて、技術者の観点からの御説明を野山様にしていただきたいと思います。どうぞよろしくお願いします。

(ｱ) 報告「Java 概説」（ゲスト講師・野山孝太郎氏）（第 2 回 資料 3「Java（ジャバ／ジャヴァ）概説」）

【野山】　改めまして、富士通の野山と申します。本日はよろしくお願いいたします。

富士通のほうで長年 Java も含めて様々なソフトウェアやプログラミングの技術支援を担当しております。Java に関しても、実は黎明期というか、もう 20 数年プログラマーとしては絡んでおります。私自身、法律などに関してはもう全くの素人ですので、飽くまでもエンジニアの視点ですが、本日話題になっている、Java API とは何物なのか、というところを少し技術視点で御説明をさせていただければと思います。よろしくお願いいたします。

（スライド 1）それでは、スライドのほうに御注目いただければと思いますが、アジェンダというほどのものではないのですが、Java API、これが何物なのかというのを御説明するために、そもそも Java というものがどういう位置付けのものなのかというところと、Java API と密接に関連してくる Java Specification Request、Java の仕様要求というものがございまして、これがいったいどういう位置付けのものなのか、それを受けて改めて Java API とは何物なのかといったところを御説明させていただければと思います。

また、冒頭の御説明にもあったとおり、今回の裁判の中で、実際のプログラミングコードが示されているところもあるので、若干中にプログラミングコードの読み方というか、後ほど API にも絡んでくるので、少し細かいプログラミングの御説明も含んでおりますが聞いていただければと思います。

a　Java とは

（スライド 3）まず Java ですが、実は、この後の Java API にも絡んでくるのですけれども、正

直なところ、エンジニアの世界ではあまり明確かつ厳密な用語定義がされないケースが多くて、習慣的に様々な意図で 1 つの単語を使うというのは、Java だけではなくていろいろなところで見られる現象なのですけれども、Java や Java API も複数の言葉で通常エンジニアが使っているケースが非常に多いかと思います。

Java というと、1 つはプログラミング言語ですね、Sun Microsystems が作って買収されて Oracle に版権が移ったプログラミング言語の名前ですし、また、例えば Windows に何かアプリケーションを入れるときに Java のインストールが必要という文脈では、Java のプログラム自身を開発したり動作させるためのプラットフォーム開発環境、実行環境を Java と呼ぶこともあって、文脈によって言語の名前だったり、プラットフォームの名前だったりと、この辺りもファジーに使っておりますが、いずれにせよ、プログラミング言語なり、それを動かす環境の名前が Java というものです。

プログラミング的な特徴はいくつかあるのですが、少し Java API につながるお話として、オブジェクト指向、プラットフォーム非依存という 2 つの観点でもう少し Java というものがどういう言語なのかを御説明させていただきたいと思います。

（スライド 4）こちらのページは本当にご参考です。今日のテーマと何ら関係ないのですけれども、数ある言語の中の 1 つ、こういう特徴を持つものだという表をご参考までに載せております。

（スライド 5）先ほどあったプログラミングコードを読むために、少し Java の設計思想を御紹介したいと思って、Java の特徴であるオブジェクト指向を少し御紹介しています。

オブジェクト指向というのは、プログラミングパラダイムと呼ばれますけれども、プログラムを作る上での設計の大きな考え方、代表的なものがいくつもあるのですが、そのうちの 1 つになり、下のほうに例を書いています。非常にシンプルな例ですが、いわゆるレンタル CD とかレンタル DVD みたいなお店のシステムですね。

それで、代表的なプログラミングパラダイムとしては、COBOL とか C 言語が採用している手続き型と呼ばれる、やらなくてはいけない処理、コンピュータで実現したい処理を時系列にというか、どんどん記述していく。例えば、レンタル DVD であれば、借りたい DVD の商品番号を入れて、もし 2 本、3 本と借りたければどんどん商品番号を入れて、会員番号を入力して、もし会員ではなくて会員番号がなければ、名前を入れて住所を入れて会員登録をして、そうすると価格が表示されてお金を払ってお釣りが出てというような処理をどんどん書いていくような考え方です。

それで、Java が採用しているオブジェクト指向というのは、実現したいプログラミングによって実現したい処理に関係するような何かしらの概念、属性と呼ばれるいわゆるデータデータや機能、この機能はほかの言語でいえば関数とか、サブルーチンのような位置付けですが、こういう機能を持つオブジェクトという概念で表現、設計をするという考え方です。

ですので、一例ですが、例えばレンタル DVD の場合ですと、会員というオブジェクトがあって、この会員オブジェクトは会員番号とか氏名とか住所というデータを持っている。あと店舗というオブジェクトがあって、店舗番号とか在庫というデータを持っていて、この店舗オブジェクトに問い合わせることで、在庫の確認とか新規入会などができるとか、こういうふうに現実世界にあるようなもの、若しくは現実世界にない抽象的なものを含めて、概念に対してデータや機能を貼り付けて、オブジェクトという単位で設計するというのがオブジェクト指向の考え方になっています。

（スライド 6）それで、このオブジェクト指向の中で、この後先ほどのサンプルのプログラミングでも出てきますが、いくつかオブジェクト指向における用語というのがあって、一番 Java のベースになるのがクラスと呼ばれる考え方ですね。このクラスというのはいわゆるオブジェクトの雛型です。

例えば、先ほどの会員オブジェクトであれば、会員が 100 万人いるとすると 100 万個のオブジェクトができるわけですが、それらのオブジェクトは共通のデータや共通の機能を持っていると。

それで、まず Java でこういうプログラミングをするときには、会員クラスのような、会員クラスというのはデータとしてこういうものを持っていて、こういう機能を持っているものだよというのをプログラムとしてきちんと実装するのですね。で、それに対して実際にデータを入力する

ような形で実体化させる。そして、この実体化されたものがオブジェクト指向におけるオブジェクトとか、又はインスタンスと呼ばれるようなものになってきます。

クラスの中のデータをフィールド、更に機能をメソッド、メソッドはほかの言語でいうと関数とかサブルーチンのようなイメージになってきますが、こういったものが定義されるというのがオブジェクト指向の基本的なプログラミングの考え方になってきます。

（スライド 7）そして、もう 1 つ、先ほど会員クラスから実際に 1、2、3 という会員番号を持ったオブジェクトを作るみたいな絵がありましたが、このクラス同士を「継承」という名前で属性や機能を引き継いで、更にプラスアルファをしたクラスを作るという設計概念があります。

ここも非常にシンプルな例ですけれども、例えば従業員というクラスを作って、その従業員クラスは従業員 ID や名前というデータを持っている。そして、この従業員クラスを引き継いで、例えば管理職という従業員の属性機能に追加してほかの属性や機能を持つクラスとか、営業部員、開発部員とか、この部署ごとに従業員を引き継いだクラスを作るというような考え方ですね。それで、例えばこの枠で開発部員のクラスに実際にデータを入れてオブジェクト化すると、親クラス、子クラスと継承の親子関係で呼びますけれども、親クラスである従業員の属性機能と子クラスの属性機能全体を持ったオブジェクトが出来上がる。このように実体の実装したいものをクラスに割り当てて、またクラス同士の親子関係も設計に入れてというのがオブジェクト指向における基本的な設計の考え方になってきます。

（スライド 8）それで、プログラミングコードっぽく実装イメージですね、先ほどの会員クラスというところに戻って、会員番号、名前、住所というデータと機能を持っていると。これをプログラミングで実装するとここの枠にあるような書き方になります。

いろいろと省略をしているので、これ単独で動くようなものではないですが、今ここで書かれているもののうち赤太字になっている部分が Java として決められている単語ですね。

Java で作るときには先ほどクラスというのが前提になるというお話をしたのですけれども、クラス〇〇、この場合だと会員（Member）というクラスを作ります。数値型の会員番号（id）、文字列型の氏名（name）、文字列型の住所（address）というデータ、フィールドですね、これを持っていますと。

また、こういう関数メソッド、機能を持っています、というのを定義して、実際にこの Member クラスを呼び出す側としては、Java の場合、new〇〇という書き方で実際にそのクラスからオブジェクト化、インスタンス化、実体化させるのですが、そういうふうにしてオブジェクトを作って、それから関数などを呼び出す。こういうプログラミングイメージで実装していくというのが Java ですね。

（スライド 9）そして、もう 1 つ Java の特徴として、プラットフォーム非依存という特徴を持っています。Java の基本思想として、Write Once, Run Anywhere という有名な言葉があるのですが、1 度プログラミングコードを書けばどんな環境でも動くよと。ほかの多くのプログラミング言語の場合、例えば Windows と Linux と Mac OS のように OS 環境が異なるときに、用意されているライブラリなどが異なっているケースであれば、ソースコードも大半は流用できるものの、そういう環境に依存したコードというのは別々に書き分けなくてはいけないと。で、このソースコードをそれぞれの環境用のコンパイラなどでコンパイルをして機械が読めるバイナリコード、機械語に変換してこれを実行するというのがほかの多くのプログラミング言語です。

Java の場合は、右側にあるように、ソースコードを書いて、それを、javac と呼ばれるコンパイラが準備されているのですが、そのコンパイラにかけたときに、ほかの言語のように一気に環境ごとの機械語に翻訳するのではなくて、汎用的な中間言語に翻訳をします。Windows、Linux、Mac OS などの環境ごとに Java の仮想マシーン、JVM と呼ばれる中間言語を介してそれぞれの環境の機械語に翻訳してくれる機構が準備されていて、この環境ごとの JVM 仮想マシーンがあると、1 度書いたコード、また 1 度コンパイルした中間言語を Windows に持っていけば Windows の機械語に翻訳しながら動かしてくれるし、Linux に持っていったらそのまま Linux で動かしてくれる。それぞれの環境でコンパイルして専用の機械語にしなくて良いという、マルチプラットフォームで動くというのが Java の 2 つ目の大きな特徴です。

（スライド 10）ここで、本日の本題の Java API の話が少しだけ出てくるのですが、この JVM は、つまりコンパイルした後に Java の中間言語を機械語に翻訳して動かす機構なのですが、実際に機械語に翻訳するような機能と、後はメモリ空間とか CPU を操作するような非常に原始的な

機械操作の部分を代行するような機能しか備わっていなくて、Java をプログラムとして成立させるような、いわゆる標準ライブラリと呼ばれるような標準機能ですね、この辺りは JVM としてはほとんど提供されていないのですね。

ですので、JVM を単独で利用するというケースは非常にまれで、Oracle もこの解釈機である JVM と、Java の標準クラスライブラリ、これが一般的に Java API と呼ばれるのですが、この JVM と Java API をパック、セットにして、JRE、 Java Runtime Environment という Java の実行環境の単位で配布をしていますので、Windows に Java の環境をインストールすると、インストールされるものは JVM ではなくて JRE ですね。JVM プラス Java API、これが基本的にはインストールされます。

更に、JRE に中間言語を作り出す Java のコンパイラとか、各種開発ツールを組み合わせた形で、JDK、Java の開発環境というセットでも配布をされています。

ですので、一般的にどこかから Java をダウンロードしてくると JRE か JDK どちらかをダウンロードしているはずなのですね。

（スライド 11）更に、Java にはこのセットの組合せと Java API の組合せかつ JVM の種類によっていくつかエディションが用意されています。

一般的に皆さんよく使われるのは、Java SE と呼ばれる、Standard Edition、一般ユーザー向けで非常に標準的なクラスライブラリが揃っているものになります。

そのほか Java EE と呼ばれる、企業ユーザー向けというか、企業システム向けというか、サーバーで大型の業務システムを作るときのためのエディションですね、Java の標準クラスライブラリの中にそういうサーバー向けの機能が加わったものであるとか。

あとは Java ME、 Micro Edition と呼ばれる組み込みシステム、携帯スマホなど向けの Java のエディションも用意されていて、ここも標準クラスライブラリの中身が異なっていたり、あと JVM 自体も SE や EE と比べて軽量化されているとか、このようなエディションがございますが、今回は Java SE に絞って、この後御説明をさせていただきたいと思います。

（スライド 12）では、改めて Java の特徴、オブジェクト指向はこういう書き方だという部分とマルチプラットフォームで動くという御説明をさせていただいた後に、今話題となっている java.lang.Math というクラスのコードの一部を持ってきております。

先ほどの御説明のとおり、Java の基本はクラスという単位で物事を書いていくので、これは Math と呼ばれる、数学でよく使われるような数値計算とかそこに役立つような機能が詰まっているクラスですが、Math というクラスを定義しますと。ここはフィールドですね、属性として E とか円周率とかそういうものが用意されていまして、後はメソッド、関数、機能としてこういったものが用意されていますというのがずらっと定義されているものになります。

それで、ここまで御説明していないものとして少し色分けしていますが、package、 import とか、public、private とか、final、 static とか、こういう Java の予約語がずらずら書かれていたり、double とか int とかいろいろと単語が書かれていますが、この後 API 仕様を読む上でもこの辺の単語が出てくるので、簡単にどういったものか触れさせていただきます。

（スライド 13）まず、int とか double といった辺りは、Java で用意されている基本的なデータの型になります。Java というのは厳密な型決めをする言語なのですね。つまり、これは数値型でこういうデータ範囲のものを扱えるよとか、これは文字列型だよとか、そういうデータの型を明確に最初に宣言をして規定するというルールの言語になっています。

それで、これは標準ライブラリではなくて、JVM の範囲で使えるデータ型として、true or false の真偽値であるとか、ABCDEFG 0123 みたいなひと文字であるとか。後は数字ですね。数字も格納できるメモリ領域というか、数字の大きさによっていくつか種類があるのですけれども。数字とか後は小数点ですね。こういったものが用意されています。

（スライド 14）それで、先ほどの java.lang.Math の中にあった package、 import のところなのですが、Java のプログラムを本格的に組み始めると、非常にたくさんのクラス、ほかの人が作った標準ライブラリであったり、若しくは拡張ライブラリであったり、非常に大量のクラスを利用するのですが、1 つのところにクラスのファイルがずらっとあると名前の重複等含めていろいろと問題があるので、Java というのはクラスファイルを Windows でいうところのフォルダのような階層構造で管理しているのですね。

それで、package〇〇というのは自分のいる場所、import〇〇というのは利用をしたいほかのク

ラスの場所を指しています。なので、先ほどの class Math の場合、package java.lang また、import java.math.BigDecimal と書かれていたのですけれども、これは Java というフォルダがあって、その下に lang というフォルダがあって、Math 自身はここにいますと。同じく、その Java フォルダの下の Math というフォルダの下に BigDecimal というクラスがいてこれを使いますよと。そういうフォルダ構造で自分の場所、使いたい別の人、呼び出したい別の人の場所、これを定義しているのが冒頭の部分ですね。

（スライド 15）また、先ほどのフィールドやメソッドのところに public、private という書き方がされていたのですが、これはアクセス修飾子と呼ばれるものです。今日 Java の細かいところまでは不要かと思いますので下の表はいちいち読み上げないですが、早い話、メソッドとかフィールドに対して誰がどうアクセスできるのかというのを規定しているのがこのアクセス修飾子ですね。この後出てくる API に関しては、基本的に外からアクセスできるからこそのインターフェースなので、API に規定されているものは基本的にはほぼほぼパブリックのもの、ほかのどのクラスからもアクセスして良いというものになります。

（スライド 16）後は final と static ですね。ここも Java の細かいところは良いかなと思うので、こういう予約語が Java にはあって、final というのはそのメソッドやフィールドが変更できないというのを表しています。それぞれの対象ごとに少し意味合いが違いますが、変更できないものというのを宣言しているものです。

（スライド 17）static というのは、これも細かく説明し始めるときりがないのですが、メソッドやフィールドの外側からの呼び出し方が変わるのですね。static が付いていないと、先ほどあったように、クラスからオブジェクトを作って、それからしか呼び出せないのですが、static がついていると、「クラス名.メソッド名」という形で直接呼び出せるというふうに呼び出し方が変わります。今日は意味については良いかなと思うのですが、static が付いているか付いていないかで、利用する側の呼び出し方、インターフェースが変わる修飾子が付いております。

（スライド 18）ですので、改めて java.lang.Math ですね。右側に書いておりますが、ここでプログラミングされていることが何を表しているかというと、私は java フォルダの下の lang フォルダの下にいますよ、java フォルダの下の math フォルダの下にある BigDecimal を使います、ほかにもこういうのを使いますよ、と。で、私自身は皆さんに公開している変更できない Math というクラスで、更に外に公開している 64 ビットの少数の定数を持っていますよ、値はこういうものですよ、と。で、更に公開している max という名前の static、呼び出し方が少し特殊なメソッドを持っていて、32 ビット整数の数字を 2 つ受け取って 32 ビット整数を返しますよと。こういうのが先ほどのプログラミングコードの中で定義をされているところですね。

b　Java Specification Request とは

（スライド 19）それで、今の java.lang.Math の話は、この後 Java API のところで戻ってくるのですが、一旦話を変えさせていただいて、Java Specification Request というものを説明させてもらいます。

（スライド 20）一般的に縮めて JSR と呼ばれるのですが、Java に関連する技術はこの Java 自身、JVM の仕様とか、Java API の仕様とか、そういったものを含めて全て JSR という名前で公式に仕様が決められています。この仕様自体は JCP という団体、Java Community　Process という団体で策定をしていて、この JCP の中に Executive Committee というところがあって、そこで意思決定、この仕様はこうしようというのを決めているのですね。で、この Executive Committee の現在のメンバーは、主要開発者というか、著作権者である Oracle のほかに、Amazon とか弊社も参加しております。また、IBM、Intel、Microsoft、アリババ等々、そういった企業のエンジニアから構成されるところで Java の仕様を策定しています。

例えば、今 Java の最新バージョンはバージョン 19 というものですが、その Java の 19 は JSR 394 というもので仕様定義をされています。

ちなみに、先ほどもちらっと言いましたが、この JSR のほとんどのもののコピーライトは Oracle になっているので、Oracle 著作ですが、一応形式的には JCP というコミュニティのようなもので、更に複数の企業からなる Executive Committee で仕様決定している、そういった仕様があります。

（スライド 21）このページも Java のバージョンがずっとある中で JSR という形で仕様策定されているというものを表にしただけのものです。

（スライド 22）ここでまた先ほどの java.lang.Math に戻ってくるのですが、例えば java.lang.Math が JSR 394 の中でどういうふうに定義をされているかを実際の定義書から抜粋したものが右側の四角になります。

御覧いただいて分かるかと思うのですけれども、ほぼほぼプログラミングコードベースというか、コードベースなのですね。これは Java の外部仕様書なのですけれども、JSR 394 の中で、例えば java.lang.Math は package java.lang の中に置いてくださいと。公開している、変更できない Math クラスですと（「public final class Math」）。Java のコードそのもので外部仕様も定義されていると。「public static final double」と、先ほどの予約語の列があって、E という定数を持ってくださいと。この E の定数にはこの数字を当てはめてくださいという形で、ほとんどプログラミングコードベースの外部仕様になっていると。なので、この外部仕様に従って java.lang.Math を組んだ瞬間にほぼほぼこの部分はこのとおりの記述です。

もちろん、この a や b は任意なので、x と y に変えるとか、そういった細かいことはできるのですが、冒頭のところの予約語の列に関しては、仕様に従う限り、ほぼほぼこの組み方しかない形になるというのが JSR として定義されているものですね。

（スライド 23）ちなみに、Java と呼ばれるものは非常に複雑怪奇でして、要求仕様というのが、先ほどあったとおり、Oracle の著作権ではあるものの、JCP というコミュニティによって一応合議制で定められているということになっています。

それで、要求仕様に従って実装されているものがいくつかあって、1 つは Oracle の Java ですね。これは当然 Oracle が自分たちの著作での要求仕様に従って Oracle として提供している Java というものです。

もう 1 つが、これもソースコードの著作権は今 Oracle なのですが、OpenJDK、Open Java と呼ばれる要求仕様に従ったオープンソース実装の Java JRE、JDK というものが存在します。で、今この著作権は Oracle ですが、GNU GPL v2.0 で配布されているものですね。

過去には、冒頭、Google もライセンスを受けようとして駄目だったという話があったのですが、Oracle から正式にライセンスを受けて各社で JDK を実装するのが流行っていたというか、みんなやっていた時期があって、IBM さんとか弊社も Oracle さんからライセンスを受けて、富士通版の JDK というのを独自で作っていたりもしました。

ただ、最近はライセンスを受けて実装というよりは、GNU GPL で配布されているオープンソース実装、これはソースコードなのでオープンソース版のソースコードを Oracle が公式にビルドしている Oracle Open JDK というものも存在するのですが、このオープンソースを有力な会社が独自にビルド若しくは機能追加する形で、Red Hat OpenJDK や Amazon Corretto という形で独自ビルトをしたものを各社のビジネスで使うケースも増えております。

本日の本題ではないかと思い、非常に簡略化して書いていますが、Oracle JDK が一時期有償化をしたときに、一般的に使っている Java の実行環境にいきなりお金を払えないということで、各社によるこのオープンソース実装の独自ビルドみたいなものが加速をして、恐らくそういう動きも受けて無償に戻しました。

つまり、Java と一言で言っても、実は Oracle 実装のものとオープンソース実装のものがあって、更にオープン実装のものはそれを Oracle が公式にビルドしたものと各社がビルドしたものというふうにバリエーションが存在していると。ただ、その元をたどっていくと、先ほどの要求仕様、JSR に従ってくるというのが、非常に複雑怪奇なのですが、Java の実行環境、開発環境の実態になっております。

（スライド 24〜27）ちなみに、JSR は JSR のサイトから見られるのですが、ダウンロードして解凍しますと見るのに非常に手間がかかるので、JSR がどういうライセンスで配布されているかというのを少し御参考までに後ろに載せております。すみません、ちょっと素人が解説するとあれなので一切解説はしないですが、こういったライセンスで配られております。コピーライトが Oracle なのと、やはりいろいろとコピーなどに関しては制限がかかるようなライセンスとして配られているようです。

c　Java API とは

（スライド 28）ということで、少し java.lang.Math の読み方かつそれらが Java 要求仕様という形で元々定義をされているというお話までしたところで、改めて本日の議題である Java API とは何

なのかというところに戻らせていただきます。
（スライド 29）API、一般的に Application Programing Interface ですね。一般論で書いていますが、ソフトウェアコンポーネント、プログラムやサービス、これらがお互いにやり取りをするときに使うためのインターフェースの仕様のことを API と呼んでいます。
ですので、関数、サブルーチン、メソッドなどの呼び出し方、名前、後は何をインプットすれば良いのか、そこから何がアウトプットされるのか、データ構造がどうなっているのか、そういったところが定義されて、API を見ればどういうふうにライブラリを呼び出せば良いのか、関数を呼び出せば良いのか、どう使えば良いのか、そういうのが分かる外部仕様ですね。これが一般的には API と呼ばれています。
（スライド 30）それで、Java における API 、Java API とは何なのかというと、冒頭でも言ったとおり、エンジニアの世界ではファジーに Java API という言葉を使っていると私は認識をしております。
例えば Java のライブラリがありますと。で、このライブラリを使いたい人、利用者の人がどういうふうにライブラリを呼び出せば良いのかというライブラリの使い方、呼び出し方ですね、このオレンジの部分が一般的な定義における API と呼ばれる部分だと思っているのですが、Java の場合、あまりこの部分を Java API という言い方はしていません。
で、ライブラリのところも、ライブラリのインターフェース、どういう外部仕様になっているかというインターフェースの部分と、そうやって呼び出された後実際に何かの動きをするためのライブラリの実装の部分、当然これが合わさって 1 つのライブラリとして配布されているのですが、一般的に Java API と呼ぶと、このインターフェースとか呼び出し方とか、そちらではなくて、このライブラリ全体、実装コードまで含めて Java API という呼ばれ方をされることが多いかなと。
で、Java の場合、このインターフェース部分だけは Java API 仕様という言い方で呼ばれていることが多いかなと。マニュアルとか、Oracle の公式の文書を見ていても、こういう使われ方をしていることが多いですね。
（スライド 31）それで、API の考え方も少し Java は特殊かなと思っていて、この左側が Google の公開している Google のサービスに対する API の仕様です。この真ん中のところ（行）が API の仕様です。一般的に我々にとっての API 仕様は、API を使いたい利用者向けのドキュメントなので、これは Google のところからそのまま持ってきているのですが、ここに書かれている API 仕様というのは利用者がほぼそのまま実装可能というか、利用可能、この部分をコピーして必要な部分を書き換えると呼び出せる、利用者にとってのインターフェースですが、こういう形で API が定義されているのが多いのではないかと。
逆に、ライブラリの開発者側、API を受けて動く開発者側は、API 仕様を紐解いて、どういうデータが付随しているのか、どういうデータが紐付いているのかとか、ここを解釈して更に実装コードを書いていくという処理が必要なのかなと思っています。
ただ、Java の場合は、先ほども御覧いただいたとおり、API 仕様として公開されているものは決して利用者向けではなくて、どちらかというとライブラリ開発者というか、Java API の開発者向けで、API として定義されているものはほぼそのままプログラミングコード、ライブラリ側のプログラミングコードのインターフェース部分に使えるのですよね。先ほどの public、final、static という予約語も含めて、ほぼほぼプログラミングコードになっているので、こちら側はほぼ実装可能ですと。
逆に、利用者側は Java の知識がないとどう呼び出せば良いかが分からなくて、例えばこの文章から java.lang という package にあるのでインポートの必要はないと、static が付いているメソッドなのでインスタンス化をする必要がない、特殊なクラスで直接呼び出す必要があると。更に int 型の数値を 2 つ渡すと int 型の戻り値がもらえるというような、プログラミングの知識のある人が読み解いて、こう呼び出せば良いというふうに利用者側の読み解きが必要ということで、API の仕様の書き方として少し Java は特殊なのかなと思っております。
（スライド 32）それで、Java の API Specification、Java の API 仕様ということで、こちらも公開されておりますが、ほとんど先ほど御紹介した JSR という要求仕様と書き方は変わっていなくて、ほぼコードのまま API 仕様という形で公開をされています。
ちなみに、先ほどから標準ライブラリとか Java API についてお話をしているのですが、（スラ

イド 33〜38）Java SE に含まれている API にどういったものがあるか、これもこの次のページからずらっと並べております。具体的なクラスではなくて、用意されているクラスをある程度まとめたパッケージやモジュールの単位でずらっと書いていて、更にこの下にいくつかのクラスが紐付いているので、非常に大量のクラスライブラリが準備されているのですけれども、モジュール単位で 3 ページにまとめております。

特に java.base と呼ばれるモジュールの中には Java を扱う上で非常に基本的な API というかライブラリが準備されていて、この java.base モジュールの中に更にその後ろ 36、37、38 の 3 ページにわたってのパッケージ、更にこのパッケージの下に複数のクラスという形になっているのですが、先ほどから例に出てきた Math クラスが存在している java.lang も java.base モジュールの中の java.lang パッケージの中にあります。

特にこの java.lang パッケージというのは、これまで御紹介したたくさんのモジュールやパッケージの中でもかなり特殊な存在で、先ほどのプログラムの例の中で、ほかのクラスを呼び出すときにはインポートということでここにあるこれを使うよと最初に宣言しなくてはいけないのですが、この java.lang の中に入っているクラスというのは Java のプログラムをやる上で本当に基礎の基礎みたいなものが入っているので、この java.lang パッケージだけほかと違ってインポートをしなくてもデフォルトで使えるようになっているのですね。

ですので、ここは本当に標準中の標準のものなのですが、それに合わせて様々なネットワーク操作とか、データベース操作とか、入出力の操作とか、そういうのをやる様々な機能が Java API 標準クラスライブラリとして準備をされているという状況です。

（スライド 39）その java.lang の中にどういったものが含まれているか、先ほど言ったとおりプログラム開発においては必須のものが多いのですが、Math 以外に、例えば冒頭のほうで継承というお話をしたのですけれども、Java でクラスを作るときには必ず継承元をずっとたどっていくとオブジェクトというクラスを継承しているのですね。Java は必ずこれを継承してほかのクラスを作っていくというルールがあって、そのオブジェクト指向の大元になるようなクラスも Java lang パッケージの中で標準ライブラリとして提供されています。

それで、企業のシステムなどではエラー処理をかなり実装しなくてはいけないのですが、このエラー処理を実装するための機構というのも、JVM の標準ではなくて標準ライブラリのほうで提供されている。なので、やはり普通に Java でプログラミング開発をするときには Java API、特にその中の java.lang 等はほぼ必須となるような機能として提供されているので、こちらも途中にあったとおり、Java は、JVM 単独ではなくて API を含めた JRE という単位で配布をされています。

それから、JCP、 JSR、途中に触れた各仕様等々の場所を最後に関連情報として URL を載せております。

はい、すみません、駆け足の説明の部分もあったかと思いますが、Java API と呼ばれるものが、どういったものなのか、またその前段として Java 、 JSR のところを御説明させていただきました。私からの一方的な説明はここで一旦切らせていただきます。

（イ） 質疑

【山神】 野山さん、とても複雑怪奇な世界をコンパクトにまとめていただきまして、ありがとうございました。

ここで質問を入れますかね。今まで聞いていたところで、少し技術的なことですので、よく分からないということがあったら一旦ここで受けた後で、更にもう 1 回、法律問題、奥邨先生と石新先生に説明していただいて、またそこでも質問していただけますけれども、我々の畑違いのところでもありますので、今のうちに押さえておきたい、確認したいということがございますでしょうか。いかがでしょうか。挙手していただいて。椙山先生どうぞ。

【椙山】 椙山です。ありがとうございました。分からないところはいっぱいあるのですが、特に 31 ページのところで、利用者と開発者というふうに言われたのがどういう意味なのか説明していただけますでしょうか。

【野山】 （スライド 31）こちらですね。まず、一般的にはライブラリの提供者は大体 1 社あるいは 1 団体です。例えば、Microsoft さんが何かのライブラリを出すときには Microsoft さんしかそのライブラリを出さないので、そのライブラリのインターフェースというのは Microsoft さんの

ライブラリを利用する人がどう使えば良いかという視点でしか書かれません。それが一般的で、例えばGoogleのAPIもそうです。Googleの APIは飽くまでもGoogleを使う人に向けたメッセージとして書かれていて、一般的な API は多分そういう考え方だろうというふうに思っています。

ただ、Javaの場合は、この前の方でも少しお話したとおり、Javaの仕様に従ってライブラリを開発する人というのが大量に現れるモデルなのですよね。Oracleだけがライブラリを作るわけではない。Oracleも作るけれども、オープンソースとして配布している実装もあって、その実装をもとにRed HatやAmazonが手を入れているものもあってというふうに、ライブラリの提供者が複数存在するようなモデルでやっていて、それを表現するかのように、確かにAPIの仕様も利用者ではなくてこのライブラリをみんなで作って広めましょうね、みんなでこのライブラリと同じようなものを作ってください、と言わんばかりの仕様の書き方になっている。ライブラリの開発者に向けたメッセージになっているというのが Java の特徴としてあるのかなという意味でここを書いております。

【椙山】　すみません、その利用者の方が特によく分からないのですけれども。

【野山】　利用者というのはライブラリを使う人、開発者ですね。ライブラリ自体の開発者が下、このライブラリを使って別のプログラムを作る開発者が上の利用者です。

これで御説明になっていますでしょうか。

【椙山】　少し分かりました。ありがとうございました。

【山神】　伊藤先生から手が挙がっているかと思いますけれども、どうぞ発言をお願いいたします。

【伊藤】　今最初に手を挙げた理由は椙山先生の質問に対する補足として私も横から口を出そうかと思っていたところだったのですが、この問題が解決されたので、別の質問をと思いまして。

同じこのページ（スライド31）にありますけれども、このAPIの仕様というところ、Javaの場合は主にライブラリの開発者向けということなのですが、実際このようにJSRの段階でそれぞれのクラス、あるいはそれぞれの関数の名前とか、引数の型、種類、数というのが全てここに書かれているように定まっているという理解でよろしいでしょうか。

【野山】　はい、おっしゃるとおりです。すみません、今画面を開いております。これが実際のJSRと呼ばれるJavaの仕様書です（仕様書が記載されたウェブブラウザ画面を投影。）。

この中でComplete API Specificationという形で規定をされているのですが、先ほど資料の後ろの方にこういうモジュールがありますというふうに書かれていたようなモジュールがずらっと並んでいるのですが、説明のために何度も今日出てきた java.lang.Math に行くと、このような形で「public final class Math」、これは本当にプログラミングコードほぼそのものの形でクラス定義がされて、クラスの説明などはあるのですが、定数、フィールドのところもプログラミングコードを書きなさいという形の定義であったり、このメソッドのところも、説明をぶらさないためにmaxで行きますが、例えば先ほどのmaxというのも「public static int max(int a, int b)」みたいな形ですね、ほぼプログラミングコードとしてJSR自体が規定されています。

【伊藤】　それで、先ほどのスライドの 31 ページに戻りますと、これを見れば、ほぼそのままライブラリの開発者にとっては実装可能だということは、これは実際、その宣言部分だけではなくて、中でaとbという引数を受け取って、どちらが大きいか比較して大きい方をリターンするというようなプログラムの中の実装部分を書くことがほぼそのままできると、ここまで細かく書いてくれればできるという意味でしょうか。

【野山】　呼出し部分はですね。はい、そうです。

【伊藤】　呼出し部分がそのままということですね。そのような意味ですね。分かりました。ありがとうございます。

【山神】　次に、谷川先生、お手を挙げていただいたので、どうぞ発言をお願いいたします。

【谷川】　谷川です。このJSRを最初に作った段階では何か参考にしたコードというのがあったのでしょうか。それとも、これから初めてコードを作っていくという段階でまず仕様から作ったのでしょうか。

【野山】　原則論としては、まず仕様から、という考え方ではあるのですけれども、基本的にはJavaももう長い歴史になってきているので、Javaの元々の実装コードから仕様を作って、それをJSRとして昇華して、それをアップデートしてというイメージですね。なので、全くの新機能などはJSRが一応先にあって、それに従ってJavaの新機能が実装されるという流れに今なっている形ですね。

【谷川】　では、この java.lang.Math でいうと、もともと Oracle が書いたコードがあって、それが仕様になったということでしょうか。基本的なクラスについては。
【野山】　そうですね、時系列でいうと、Sun Microsystems が書いたコードがあるのですが。
【谷川】　著作権も元のコードの著作権が承継といいますか、あるということですかね。
【野山】　そうですね、ですので、今 JSR、またオープンソース実装としての Open JDK、ほとんどのファイルのコピーライトは Oracle になっていますね。
【谷川】　ありがとうございました。
【山神】　次に、片山先生、どうぞ御発言ください。
【片山】　すみません、片山です。御説明ありがとうございます。大変勉強になります。
　今の JSR のところで、基本的なところで大変申し訳ないのですけれども、JSR で規定しているのは既に標準クラスみたいな出来上がっているものの仕様が決まっているという理解で良いでしょうか。どういう意味かというと、JSR でこういうようなクラス名で何か作りなさいとか、そういうことを何か決めているというのではなくて、今存在しているクラスの仕様はこうですよと言っているという理解で良いのでしょうか。
【野山】　例えば、Java の次のバージョンで標準クラスとして新しいクラスを追加したいというようなときには、まず JSR で規定がされます。
【片山】　そうすると、新しいクラスを作るときには必ず JSR を基本的には通しなさいと。
【野山】　はい、おっしゃるとおりです。そういう形に今なっています。
【片山】　単純に分かっていないのが、クラス名とかメソッド名とか java.lang.Math という名前があって、こういう構造が作られているのが、JSR で規定されているという言い方をして正しいですか。そこが少し私は混乱しているのですが。
【野山】　JSR で規定されているという言い方で正しいですね。
【片山】　ありがとうございました。
【山神】　今の話ですけれども、恐らく、要するに誰が仕様をコントロールしているかというと JSR で、多分 Oracle が全部コントロールしてやっていくけれども、オープンに展開することもできますよという、そこら辺は手綱を引き締めながらやっているのだと思うのですよ。だから、最終的な、必ず 1 個は Oracle が著作権を持ったものが出来上がって、ただ、それとはまた違うものが出来上がる可能性もあるという理解ですかね。
【野山】　すみません、誤解を恐れずにというか、飽くまでも一エンジニアとしての見解としては、著作権とか、ほぼほぼの決定権は Oracle が握っている状態ではありますけれど、やはりこれだけ広まった Java なので、Oracle としては 1 社独占ではなくて、飽くまで富士通とか IBM とかを含めた合議制でやっているという形を取りたくて JCP という団体を立ち上げて、JSR という要求仕様に従ってオープンにやっているという姿勢を示しているのだと思っています。
　ただ、やはり JSR 自体のライセンス条件などを見ると、やはりもう Oracle の持ち物でしかないような、著作権もそうですし、利用範囲等も含めて、JSR を元にみんなが自由に何かを作って良いというようなライセンスではないように読めるので、オープンな姿勢を示しつつ、しかし、やはり Oracle が要求仕様を牛耳ってライセンシング等を含めてコントロールしているように、エンジニアからは見えています。
【山神】　ありがとうございます。私もそういう理解でしたので大変心強く思いました。すみません、奥邨先生、割り込んでしまいました。奥邨先生、それから岩原先生と続いて御発言願いたいと思います。お願いいたします。
【奥邨】　どうもありがとうございました。
　標準クラスライブラリというものの位置付けを整理して理解しておきたいなと思います。なかなか難しいのですが、例えば、たとえが余り良くはないと思いますが、先ほどオブジェクト指向の話が出ていましたけれども、著作権の議論でプログラムというと、実は、手続型言語のところの議論でほぼ終わってしまっていて、今余りオブジェクト指向とかには対応できていないのですね。ですので、どちらかというと、昔々、いろいろ議論されていた BASIC であるとか COBOL であるとか、そういうものに例えた方が分かりやすいところがあるので、もちろんたとえが間違っていれば言っていただきたいのですが。
　標準クラスライブラリというのは、例えば BASIC なり、C 言語でも何でも良いのですが、言語を動かすインタープリタなり何なりのプログラムの一部というふうに考えてよろしいのでしょ

うか。というのは、標準クラスライブラリを呼び出すというのは、BASIC なり C 言語なりの命令語を呼び出すのと同じことで、その命令語を実際に動かしている中身というのが標準クラスライブラリであると、そういうふうな理解でよろしいのでしょうか。

標準クラスライブラリというのは何かな、というのが少し分かりづらくて。そこを教えていただければと思います。

【野山】　ここも一エンジニアとしての見解でしかありませんけれども、限りなくイエスだと思っています。厳密に言うと、標準クラスライブラリとして準備されていなくても、標準クラスライブラリを一切使わずに Java のコードを書いて、コンパイルをかけて中間言語を JVM 単独で動かすというのは、もちろん、できることはできます。必ず標準クラスライブラリがないと JVM 自体が動かないというふうな形ではありません。

ただ、Java として基本的だと思われる、後ろの方にあった例えばエラー処理であるとか、例えば文字列、人の名前のような文字の集合体とか、こういったものも一切扱えなくなるので、そこも全て、自分で何か原始的なコードを組み込んでということであれば、もちろん単独で動かすこともできますが、やはり、標準クラスライブラリありきでプログラミングコードを書くのが非常に一般的な考え方なので、先ほど配布形態の話もありましたが、JVM プラス標準クラスライブラリの JRE というのが Java の最低限の実行環境だという認識が、広く浸透しているのではないかなと思います。

【奥邨】　そうするとやはり、例えば先ほどありましたが、“max”とか、ああいうものを使わないということになると、自分でいちいち、「どちらが大きいですか？」というプログラムを自分で一から書くということまですれば、標準ライブラリを使わなくても済むけれども、そうでなければ全部一から書かないといけないと。あるかどうかわかりませんが、例えば円を描くというようなクラスがあったとすれば、円を描くというクラスを使わないとすると、自分でいちいちサイン、コサイン、タンジェントを計算して、X 座標 Y 座標を計算して、円を描くしかないと。逆に言うと、circle クラスとか max クラスというのは、Java を使う普通の人から見れば、Java の言語の命令の一部だし、それを実現しているプログラム本体だというふうに考えてよろしいでしょうか。

【野山】　その考え方で差し支えないと思います。max は数字を比べるだけではあるのですが、（スライド 13）例えば Java の標準ライブラリを使わない基本データ型だと、いわゆるユニコード一文字しか使えないので、例えば Java で、私、野山孝太郎という名前を扱おうとすると、この一文字のユニコードを組み合わせて・・・といったことで、文字列操作すら非常に長文のコードを書かなければいけないと。ただ、標準ライブラリを使うと、その中に string というクラスがあって、文字列の様々な操作とか、文字列の格納ができるというようなことなので、やはり基本的には標準ライブラリなしに何かを作るというのは、現実的ではないという認識ですね。

【奥邨】　ありがとうございます。

【山神】　すみません、今の点、また岩原先生の前に割り込ませてください。

スライド 10 ページに戻していただいて。今の奥邨先生の御質問で、Java も細かいことを言えば、手続型っぽく使うこともできますという話がありますけれども、恐らく昔の例えば Basic とかそういった古いタイプの手続型の言語で言えば、この図で言うと、この JVM と Java の標準クラスライブラリが横に 2 つに今分かれていますけれども、これを 1 つにがっちゃんこしてしまったものが多分標準的な Basic のインタープリタやコンパイラとかになるのだと思うのですね。それが Java の場合はきれいに分かれていて、分離されているけれども、基本的には標準クラスライブラリがセットされている。多分そういう理解ですよね。

【野山】　そうですね。ですので、（スライド 10）この 2 つのセル（JVM と Java 標準クラスライブラリ）を合わせたものが JRE と呼ばれるもので、基本的には今どこかから Java をダウンロードすると必ずこの JRE という単位が最低単位としてダウンロードされてくる。中を見ると確かに JVM の部分と Java の標準クラスライブラリの部分があることはあるのですが、やはりこれはもうセットで配布されていますし、やはり Java の最低限の実行環境というと、もう、このまとまりだという認識で皆いると思っています。

【山神】　ありがとうございます。すみません岩原先生、割り込んでしまいまして。どうぞ御発言をお願いいたします。

【岩原】　ありがとうございます。岩原です。大変分かりやすくて、ありがとうございました。

自分の理解等含めて、間違っているかどうか御指摘いただければと思います。

先ほど少しお話のあった手続型、従来型のものとオブジェクト指向型の話でまず確認です。

共通的なことをいちいち書いていると面倒くさいから、いろいろな計算的なものはまとめてやっておくという、そういう話が出ましたけれども、手続型だと、それは単純にライブラリ、今一般的に言われる、C だったら stdio.h とか、そういうものだということでいくと違いは何もないとは思いますが、オブジェクト指向の場合は、もう完全にオブジェクトという形にひも付けて、属性とメソッドをくっつけているという意味では、ある意味、自由度を狭めて、「これはこういう形で使いなさい。」、「車は走る、止まるしかない。」といった枠組みを作っているという意味で、従来、C とか手続型言語は非常に自由度が高いというか、単なるルールだけ決めているという感じだと思うのですが、オブジェクト指向の場合は、どういう使い方をするかという世界を構築して、部品とか考え方を最初に作ってしまって、その考え方にのっとって作ると非常にプログラミングがシンプルに分かりやすくできるという、そういう感覚でよろしいのでしょうか。

【野山】　おっしゃるとおりです。飽くまでも上に書いているとおり、設計の考え方の 1 つでしかないので、もちろん Java を使って手続型言語的に記述をするというのも可能ではあるのですけれど、このオブジェクト指向という考え方に従ってやることで、例えば自分以外のオブジェクトを隠蔽化というかカプセル化、中が見えないようにして、インターフェースの部分だけが見えることで、例えば先ほど画像の例を出していただいていましたが、サイン、コサインとか、面倒くさいこととか、円を描くようなものはなく、「円を描いて。」という命令だけを送れば、中で何がやられているかは分からないが取りあえず円が描かれる、というふうな隠蔽ですね。これも手続型で、関数とかでどんどん隠蔽をしていけば、似たようなことはできるのですが、それをオブジェクトという概念とひもづけて、という設計思想になっているというふうな捉え方です。

【岩原】　そうしますと、プログラミングをしていくとき、上位概念的に、どういうことをしたいと。銀行の何か、ATM みたいなものを作りたい、そのために必要な機能、その一つの機能を実現するために、細かい機能、というようにどんどん上位から下位に分解していって、最後、内部設計が終わったところを最終的に書けば済むぐらいまでいってコーディングしていくというような感じだと思うのですが、JSR 自体が開発者がそのまま書けば良いぐらいまで落ちているというのは、そこまで含めて具体化した世界をもうオブジェクト指向の頭の方の仕様で決め切ってしまっていると。もちろん、それにのっとらないで、先ほどおっしゃったように、手続型的に単純なルールも規定されていますから、プログラミング言語的なルールにのっとって自由に書くこともできるけれども、Java の世界観で作ったいろいろな部品とか考え方とかいうものに基づけば簡単にできると。それがかなり詳細なところまで決め切ってしまっているというような理解でよろしいでしょうか。

【野山】　（スライド 23）それに関してはノーですね。それに関しては違っていて、JSR では、Java VM の仕様と、後は Java が標準ライブラリとして準備している標準機能の部分の規定しかされていないのですね。なので、例えば銀行のシステムを作るときに、業務的にどうこうというふうな処理に関しては、当然、JSR の規定とか JSR の中でこういうふうにオブジェクト指向をやると良いよというふうな規定も一切なくて、そこは各社各開発者の知見とか経験値に沿って、銀行のシステムとはこうあるべきだ、というふうなところをやっていくと。ただその中で、金額計算とか様々な数値計算みたいなところに関しては、JSR の中で利用できる標準機能、標準ライブラリが備わっているので、それを呼び出すことで、税率の計算だとか、そういったところにいろいろ簡単に利用できると、そういう形ですね。

【岩原】　はい、僕もそういうつもりで言いました。すみません、誤解があって。銀行システム側の作りたいことの仕様が決まっているという意味ではなくて、銀行システムをこれから開発者が作りたいと思ったときに、それを作るための部品が Java で用意されていて、計算の仕方とかの細かい部品の作り方がもともと予定されている、それが仕様で決められていて、その中身は本来自由に決めれば良いところが思っている以上に通常よりももっと細かく書かれているから、各社の、今画面に出ている（スライド 23）Red Hat OpenJDK といったものが、そこに書かれている単なるインターフェース部分を超えて、こんなものができますよ、メソッドはこんなことが実行できる内容ですよ、とかなり細かく書かれているので、単純にそれをそのまま書けば済んでしまうと、そういう理解ですか、という意味です。

【野山】　であれば、はい、そういうイメージです。失礼しました。

【岩原】　そうしますと、結局 JSR で決めている仕様というのが、Red Hat OpenJDK とか Amazon

Corretto とかで書かれている最終的なコードになる部分と、かなりニアリーイコールなものに近いものまで JSR で決めてしまっている、ということですか。もちろんコードを書く人の癖などによってある程度の自由度はあると思いますが、ほとんど変わらないというような位置付けのところまでいってしまっている、ということなのでしょうか。

【野山】　まずインターフェースの部分ですね。インターフェースと言っているのは、例えば先ほどの java.lang.Math で言えば（スライド 22）、クラスの定義であるとか、フィールドの定義であるとか、メソッドの呼出しというか、メソッド自体の定義。この「return (a >=b) ? ・・・」というのは、メソッドの機能の中身の実装なので、ここの呼出し部分の定義（public static int max (int a, intb)）に関しては、ほぼほぼ同様のものになります。といいますか、ここからずらしてしまうと、同じ方式で呼び出せなくなるのですよね。例えば「static」という文字を外してしまうと、呼び出し方が変わってしまうので、インターフェースを合わせるためには、もう、ほぼほぼこの書き方をするしかありません。

　ただ、この中身の実装の部分に関しては、「a と b を比べて大きい方を返してね。」というのが定義されていまして、これは非常に単純な例ではあるのですが、とはいえ、プログラムの書き方的に、「a と b を比べて大きい方を返す」というのには何種類か書き方があるので、その部分は各社、各実装者に任されていますが、この呼出し部分、インターフェースの部分に関しては、もう、ほぼほぼ同じ形にならざるを得ないという形です。

【岩原】　では中身は、先ほどの仕様書に、今右の方の、「public static int max・・・」の下の辺りの文章が、「これはこういう機能を入れなさい。」ということが説明で書かれているわけですよね。

【野山】　そうです。

【岩原】　これをこのまま記述していくときに、a が b 以上と書くか、a が b 以下のときに a と b の後ろの方を反対にするかというのは自由度があると思うのですけれども、ほぼほぼある程度似ている形で書けるということなのでしょうか。単純に、今言われた、上の「public static int max・・・」というインターフェース周り、呼出し名とか中の引数とか戻り値とかの型というのは、通常のライブラリでも同じような指定をするのではないかなと思ったのですが、そういう意味では Java とほかのライブラリの決め方で何が違うのかなというのがよく分からなかったのですが。

【野山】　おっしゃるとおりで、インターフェースのところの規定は、基本的には当然固定でないと利用するがわが同じ手順で呼び出せないので、この部分はほぼほぼ固定になると思います。

　実装の部分は、すみません、今例で出しているのが、2 つの数字で大きい方というので、誰が実装してもそんなにバリエーションがないかと思うのですけれども、やはりもう少し複雑なメソッド、中でもう少し複雑な計算をして返すとか、処理をするようなものだと、やはりその部分に関しては右側にあるような、こういう処理をして返せというふうな文章しか規定されていないので、そこはかなり実装が異なってくるケースがありますね。

【岩原】　はい、分かりました。ありがとうございました。

【山神】　奥邨先生、手を挙げられていますね。どうぞ。

【奥邨】　私も今のところの関係ですけれども、例が、大きい方を返せ、ですから、この文章のまま書くしかないし、ほかに書きようがないということになるのですけれども、先ほどいろいろおっしゃったように、長い文字列を扱うとか文字列を途中で切るとか、いろいろ長い処理があって、それだと、文字列を、例えば 2 つに分割しなさい、真ん中で分割しなさいと書いてあっても、どうやって真ん中で分割するかというのは、いろいろな書き方があるということだろうと思うのですけれども、全体の仕組みの中でいうと、いろいろな書き方があるようなものの方がどちらかというと多いというような感じになるのか、それとも、誰が書いても同じようになるようなものがほとんどで、実装のところでバリエーションが出そうなものは余りないということなのか、どんな感じなのでしょうか。

【野山】　実装のところでばらつきが出るものの方が圧倒的に多いですね。

【奥邨】　そうすると、どちらが大きいかを比べるというのは例として出ていますが、こういうのはほぼマイナーなもので、無視して良いぐらいで、実際はバリエーションが出るものがほとんどだと思っておいた方がよろしいでしょうか。

【野山】　そうですね。そういう認識でよろしいかと思います。

【奥邨】　はい、ありがとうございます。

【山神】　野山さん、ありがとうございました。
【野山】　こちらこそどうもありがとうございました。

イ　Google v. Oracle 事件の概要

(ｱ)　報告「連邦最高裁判決の概要について」（奥邨委員）（第 2 回　資料 1「Google v. Oracle 事件最高裁判決」）

a　事案の概要

【奥邨】　（スライド 1）それでは、事案の概要を御説明します。事案の技術的なところは後で御説明いただけるということですので、訴訟の経緯のところを中心に御紹介をしたいと思います。

（スライド 2）訴訟の経緯ということで、私の資料でこういう形で整理をしております。これに尽きるわけですけれども、非常に分かりにくくなっていますので、次のスライド（スライド 3）で一応整理を分けております。

（スライド 2）そこに行く前に、そもそもですけれども、まず Sun が Java を開発いたします。ただ、Sun 自体は 2010 年に Oracle に買収されますので、最終的な裁判の原告は Oracle になっております。それで、2005 年に Google が Android の開発を開始したわけですが、その際に Android を Java を使って作ろうということで Java のライセンスを受けようとするのですが、頓挫してしまいます。そこで、全く新しい言語で作るのではなく、ライセンスを受けていないけれども、Java API の一部を利用して Android プラットフォームを開発するということを Google が行ったことになります。

（スライド 26）ここについては、詳しいことはまた後の技術のところで御説明がありますけれども、概要だけ申し上げておきますと、Android の API というのは全部で 166 パッケージあるとされています。166 が全体で水色の部分なのですけれども、そのうちの 37 パッケージ、ピンク色の部分を Google は使ったということになります。それで、Android を実現するためにはこの 37 のピンク色のパッケージだけでは駄目で、別途 Google が独自に 131 のパッケージを開発しているということであります。こちらの 131 の方（緑の部分）は全く独自に開発したもので、機能的にも異なるものですので、この緑の部分は問題になっていません。この 37 のパッケージが問題になっているということになります。

（スライド 27）ただ、37 パッケージも丸々コピーして使ったということではなくて、これも訴訟の中に出てくるダイアグラムですけれども、この中で Google が使ったのは API の中の Declaration Code と呼ばれる部分で、Oracle のものをそのまま使った。Implementing Code、実装コードと呼ばれる部分は Google が独自に書き起こしたと説明されています。

（スライド 29）この辺は後で御説明いただくところに関わります。これが API の実現するプログラムの中身ですけれども、このうち黄色いところが宣言コードで丸々使った部分と Google がコピーして使ったところということになります。水色のところ、これは実装コード部分ですけれども、ここは Google が一から自分で独自に書いたということになります。

したがって、37 のパッケージに関する黄色のコード、宣言コードの部分を丸々利用していることが著作権侵害になるかどうか争われたのが本件で、2005 年にそのような形で開発したところ、Oracle が Google を提訴する、この提訴のときに著作権侵害及び特許権侵害で提訴するということになります。

（スライド 3）こちらの資料で説明をしていきますけれども、陪審は、著作物性を肯定し、著作権侵害も肯定し、特許権侵害は否定するという結論になりました。ただ、判事が陪審の判断は間違っているということで著作物性を否定するという判断を下します。

そして、これを不服として Oracle 側が CAFC に控訴するということになります。本来この地裁はカリフォルニアの地裁ですので第 9 巡回区控訴裁判所に控訴されるべきものなのですけれども、特許権侵害訴訟が論点として入っておりまして、特許権が論点に入っている場合は、控訴内容に特許が入っていなくても元々の事案として特許権に関する侵害訴訟が入っていると CAFC が subject matter jurisdiction、事物管轄権を持つということで CAFC に控訴が行われたということになります。

そこで、ふだんは著作権の判断をしない CAFC が著作物性を肯定するという判断をします。API のコピーされた先ほどの黄色いコード、宣言コードの部分ですが、これは著作物だと CAFC

は判断をし、これに対して裁量上訴の申立てがなされましたけれども、これは認められませんでした。

著作物性が肯定されたので、今度は別途フェアユースが成立するのかどうかということが論点になりました。陪審は、黄色い部分、宣言コードは著作物になるけれども、それを今回 Google が使ったのはフェアユースだと判断し、これに対して Oracle が控訴したところ、CAFC はフェアユースを否定したということになります。

それに対して、今回 Google が裁量上訴の申立てを行って、最高裁がそれを受けた当初、著作物性とフェアユースの成立の両方について判断の対象にするということで裁量上訴の申立てを受けたということになります。

（スライド 2）そして、それを受けて今回判断が示されたのが 2021 年の最高裁判決ということになります。後でまた見ていきますけれども、最高裁は著作物性についての判断は回避して、フェアユースの判断だけを行ったということになります。

b　判決の概要

【奥邨】　（スライド 11）判決の内容としまして、法廷意見の方は、先ほど申し上げましたように、最高裁は著作物性と、それが仮に著作物だったとした場合に、API の宣言コードの部分を丸々コピーすることがフェアユースになるかどうか、という 2 つの問題があったわけですけれども、最高裁は、この API の宣言コードの方の著作物性の判断は回避した、ということになります。なぜなら、仮にこれが著作物だったとしても、フェアユースが成立するので、Oracle 側は勝てない、ということですね。Google には著作権侵害の責任がないと。したがって、フェアユースが成立するから、著作物かどうかは深く議論する必要はない、という判断になっております。6 人の判事が法廷意見に賛成をしていると。反対意見としては 2 人で、こちらはそもそも著作性があるということをはっきりした上で、しかし、フェアユースも成立しないという結論になっております。

（スライド 12）それで、総論としましては、まず①Java API の宣言コードとその体系を組む全体の著作物性と、それから②Google による宣言コードとその体系の利用がフェアユースとなるかという、この 2 つの問題については、先ほど申し上げたように①のところは著作物性がある、ということを仮定した上で、フェアユースの成立の可否だけを判断する、ということを行っております。

（スライド 13）フェアユースの検討については、フェアユースには 4 要素あるわけですが、この 4 要素、通常は、法定の順番で、第 1 要素、第 2 要素、第 3 要素、第 4 要素の順番に検討されるのがほとんどのケースということになります。しかし、今回、裁判所は第 2 要素、利用される著作物の性質というところを最初に検討するということを行っております。

この第 2 要素、利用される著作物の性質というところで、これはフェアユースに有利という判断をしております。「宣言コードは、コンピュータプログラムの一部であるが、タスクを分割するという一般的なシステム、タスクを体系化するというアイデア、コマンドの呼び出し方、実装コードと、密接に結びついているという点で特徴的」だと。「宣言コードの場合は、プログラマーが、宣言コードの名前を覚えやすくする点に創造性が発揮され」ているにすぎない。「Sun の Java API は、ユーザーインターフェースであり、宣言コードは、ユーザーインターフェースの一部であるため、他のプログラムとは異なる。宣言コードの価値は、プログラマー」、このプログラマーというのは宣言コードを書く人ではありません。先ほどでいえば、Java によってプログラムを作る、アプリケーションのプログラマーです。そのプログラマー「が時間と労力を費やして API システムを習得したことに由来する」。先ほどの御説明の中でいえば、「ユーザー」と呼ばれていた部分がこのプログラマーに当たります。「以上から、仮に著作権があるとしても、宣言コードは著作権保護の中核から遠い存在であるといえる」。著作権が保護する中核になるような著作物ではない、というふうに言って、したがって、第 2 要素は著作物の性格としてはフェアユースの成立に有利な形で解釈される、というふうに言っています。

（スライド 14）次に第 1 要素、利用の目的のところですけれども、これについてもフェアユースに有利という判断をしております。通常第 1 要素の検討においては、transformative、変容力のある利用か否かというのを重視してきたわけですけれども、今回、Google が宣言コードを複製しましたが、これは機能を呼び出すという目的自体は、プログラムの複製全てに共通するものなので、より具体的な目的を考えないと、本件の検討はできないと。「Google が Java API を利用した

のは、スマートフォン向けの創造的で革新的なプラットフォームを生み出すためであった」、つまり、再実装のために必要な範囲で API を複製したにすぎないと。「インターフェースの再実装が、コンピュータプログラムの開発を促進することは、十分に陪審員に対して示されている」と。よって、こういう状況であるので、Google の複製の目的と性格には変容力がある、transformative である、という判断をしています。

今回、Google は、商業目的で行っていますけれども、しかし商業的であるということ自体は、それだけでフェアユースを完全に否定するわけではない。transformative な場合は、商業的であっても、フェアユースが認められることがあると。

あと、Google は、1 回ライセンスを受けようとしていました。ライセンスを受けようとしたということは、実はライセンスを受けないと著作権侵害になるというふうに思っていたということを意味するわけですね。しかし、Google がその時点でそういうふうに思っていたということ自体は、フェアユースの判断には別段影響は与えないと、そういうふうに言っています。

（スライド 15）それから、第 3 要素。コピーされた部分の質と量というのが問題になります。Google は 37 パッケージの宣言コード全てを複製しています。これが 11,500 行ということになります。Google が使おうと思った宣言コードは全部使っているということになります。ただ、一方で、Java の API 全体というのは使わなかったところも含めると、使おうと思わなかった 37 パッケージ以外のパッケージも含めると、286 万行になるので、複製されたのは全体の中の 0.4%にすぎない。「複製された部分が原著作物の創造的表現の核心である場合は、少量の複製であってもフェア・ユースとは言えないとする一方で、大量の複製でも、複製された部分が余り創造的表現を含まない場合や、正当な目的に必須の複製の場合は、フェア・ユースに該当する」と以前から当裁判所は言ってきたと。Google は今回、インターフェースの再実装という変容力のある目的のために複製を行った。この複製の量というのは有効で、かつ変容力のある目的と結びついたものなので、例で言えば大量の複製になったとしても創造的なものを含んでいないとか、正当な目的に必須の場合というものに当てはまりますので、フェアユースに有利に働く、というふうに言っています。

（スライド 16）最後、著作物の市場や潜在的な市場、現実の市場や潜在的な市場に与える影響ということですけれども、これについては、権利者が損失を被るか否かだけではなくて、その原因も考慮すべきだ、というふうに言っています。

例えば辛辣なパロディによる原作品の売上げの減少というのは著作権法で保護される損害ではないと。著作権が保護する価値に対する損害があるかどうかというのを見るべきだと。権利者の損害と複製が公共の利益に与える影響との比較考量も必要だと。

陪審は、Java の現実の市場又は潜在的な市場に損害を与えていないことを認定することが可能でしたし、Google の API の一部を複製したかどうかに関わらず、Google が今回やらなかったとしても、Sun 自体はスマートフォンの市場に参入できなかったでしょうと。

潜在的な市場というのは、Sun にとって理論的に可能だった全ての市場を指すというわけではないと。そうすると、あらゆる場合に損失が概念されてしまうから、そういうふうに考えてはいけない、現実性のある市場を考えるべきだと。

それで、「複製の結果、Google は Android プラットフォームからばく大な利益を得たが、その源泉は、プログラマー」、これはアプリケーションのプログラムを書くプログラマーで、先ほどの御説明では、ユーザーが、Java への投資、すなわち Java の使い方を覚えるために投資した部分に、大きく関連する一方で、「Sun（＝Oracle）が Java API を作成するために行った投資との関係は薄い」のだと。したがって、この利益の源泉がどこにあるかという点では、ユーザーの方に源泉があるのだと。だから、Oracle がこの利益の分け前にあずかろうというのは合理的な理由がないのだと。しかもここで今回の複製を認めないということになると、公衆に害を及ぼすと。Java の API の使い方を覚えた人たちが、それが全く無駄になってしまうという意味において公衆に害を及ぼすと。したがって、これもフェアユースに有利だと。

（スライド 17）4 要素を総合考慮した結果として、「Google は、ユーザーが蓄積した能力を、新しくかつ変容力のあるプログラムに生かすために必要なものだけを抽出して、ユーザーインターフェースを再実装したのであるから、Google による Java API の複製は、法律問題としてフェア・ユースである」と言って、連邦巡回控訴裁の判断を取り消して差し戻したというのが、今回の事件の大きな流れということになります。

（イ） 報告「著作物性に関する下級審の判断について」（石新委員）（第 2 回 資料 2「Google v. Oracle 事件の経過」）

【石新】　では簡単に。この資料は去年私がソフトウェア情報センターのセミナーで参加させていただいたときに、事件の経緯ということを御説明させていただいたときに使わせていただいたもので、御参加いただいた方には、御覧いただいた方もいらっしゃるかもしれません。ただ、一部は修正等して御提供しておりますが、大部分は奥邨先生から御説明があった内容と重複していますので、時間も限られていますし、私に割り当てられた著作物性の、最高裁は判断を回避したわけですけれども、下級審では判断が出ていますので、それを御紹介するというところだけさせていただきたいと思います。

a　著作物性とフェアユースの各論点で地裁と控訴審の判断が割れた

（スライド 4）先ほど奥邨先生の方からも御説明があったのですけれども、この事件、行ったり来たりしていまして、第 1 ラウンドということで、2015 年の段階で最高裁で一度不受理というケースになっている著作物性の争いです。

不受理になったのは、著作物性の争いの方の部分のケースで、今御覧いただいているページ（スライド 4）の上の部分に該当する部分で、カリフォルニア連邦地裁の方では API、何が Java API なのかというのは、先ほどの御説明を踏まえて、いろいろ難しいところがあるかと思いますが、簡単にここで整理させていただくと、API は著作権では保護されないというのが連邦地裁の判断でありました。それに対して CAFC は、いや、API は著作権で保護される、ということで、第 2 ラウンドのフェアユースの抗弁の議論に動いていったというのが、大きな流れでございます。

b　第 1 ラウンドの争点

（スライド 7）今申し上げました、第 1 ラウンドが、著作物性が争点になったということなのですけれども、事実関係については争いがなくて、すみませんここ（スライド 7）、先ほど 166 なのか 160 なのか、奥邨先生のお話をお聞きしながら自分が間違えているのかなと思って聞いていたのですが、私の誤記かもしれませんが、それは御容赦いただきまして、37 の宣言コードの名称とその体系を複製したという事実と、Google 自身が実装コードは自分で書いたという事実については全く争いがない。争点となったのは、コピーをした宣言コード、declaring code の名称及びその体系部分について、そのコピーが著作権を侵害するのか、すなわちその declaring code とその体系が著作物性を有するのかというところ。そこが争点になったわけですが、これがなぜ争点になるのかというのは、アメリカ著作権法上の 102 条の b 項という条項に基づいております。

（スライド 8）102 条の b 項の英文をそのままコピーさせていただいて赤字で書かせていただいています。

（スライド 9〜10）日本語は著作権情報センターの翻訳を利用させていただきましたが、「いかなる場合にも、著作者が作成した創作的な著作物に対する著作権による保護は、着想、手順、プロセス、方式、操作方法、概念、原理又は発見（これらが著作物において記述され、説明され、描写され、又は収録される形式の如何を問わない）には及ばない」と。操作方法、method of operation には及ばないと規定されていて、API というのは、この method of operation に当たるのだ、したがって、102 条 b 項によって著作権侵害は成立しない、というのが Google の主張でした。

（スライド 11〜）これに対して、著作物性が問題になっているということで、これに尽きてはいませんが、中心的なところだけ簡単に整理して表にまとめてみたものがあります。連邦地裁は先ほど申し上げたように、宣言コードの部分について、102 条 b 項によって著作物性は認められない、著作権は認められないというふうに判断したわけですが、「ルールに従う以上、プログラマーは同じ機能を指定する方法を示すために、同一の declaration／header を利用しなければならない。Java パッケージのそれぞれに対する declaring code を書く方法は一つしかない、よって、Merger 理論により、著作権性が認められない」と。Merger 理論、102 条 b 項に基づいてということなのですけれども、著作物性が認められない、というふうに判断いたしました。

その他 2 つ、ショートフレーズの抗弁とありふれた表現の抗弁というのが、Google から両方とも出されておりまして、それについても判断をしました。ショートフレーズの抗弁については認めまして、問題になっている宣言コードというのが非常に短くて、そこに著作物性を認めること

ができないという理由でも著作物を否定しました。ただ、最後のありふれた表現の抗弁というのは連邦地裁は認めず、37 の Java API のパッケージのグルーピングとか、コード名称がありふれたものであるという主張を支える証拠が出されていないという理由で、その点の抗弁は認めませんでした。ただ、冒頭申し上げましたとおり、プログラマーとしては、インターフェースを実現するためにはどうしても Java のルールに従って宣言コードを複製するしかない。よって、102 条 b 項によって、その部分については著作物性がない、というふうに判断したものです。

これに対して CAFC は判断を覆しまして、宣言コードにも著作物性があるのだということを認めたものです。簡単に申し上げますと、飽くまでも、102 条 b 項というのはアイデアと表現の 2 分論を規定しただけなのだから、表現する方法が複数あれば、その中から 1 つ選んだということが言えれば、それで 102 条 b 項の制限というのはクリアされて、著作物性は肯定されるべきだと。したがって、2 つ目のところで記載しておりますけれども、7,000 に及ぶラインの選択と配列というのは無制限にもともとあり得る。先ほどから話題になっております、math.. の表記、java.lang.Math.max というコード名称は別にそうでなければいけなかったというわけではなくて、そういう表現を、もともと Sun Microsystems だと思いますが、それが選んだ。ただ、ほかの表現だってあったのだと。複数の選択の余地があったというところに創作性を見いだして、102 条 b 項の適用を否定したと。Google が強く主張して連邦地裁もそれは認めたわけですけれど、Google がそのインターフェースを実現しようとしたら、そのルールに従ってコピーせざるを得ない、それ以外の選択はないのだという主張に対しては、ほかの選択があるかどうかというのは、Google が複製しようとしたタイミングで判断するのではなくて、そのコンピュータプログラムが創作されたときに判断されるべきだと。判断時が間違っているという指摘をしまして、したがって、そもそも宣言コードを創作したときには、先ほど申し上げたように、幾つも表現方法があったはずだということで、その点の Google の主張も退けたということです。

ショートフレーズの抗弁に対しては、一つ一つを捉えれば短いかもしれませんけれども、それが 7,000 にも及ぶということで、CAFC の判断を見る限り、日本法でいうところの編集著作物的な形で、いろいろ小さなものをいっぱい集積することによって、全体を捉えれば創作性が十分認められるから、ショートフレーズの抗弁は認められないと。

それから、ありふれた表現の抗弁というのは地裁も認めなかったわけですけれども、それに加えてやはり CAFC も認めず、この点でも、どの段階でありふれているかどうかということを判断するべきか、コピーしようとしているときではなくて、創作時にこの書き方がありふれていたかといえば、他にも幾らでもあったではないか、ということを指摘して、判断したということになります。

先ほど奥邨先生のお話にもありましたが、最高裁の反対意見の方では CAFC と同じような判断をしまして、飽くまでも、表現方法が幾つもあったということが言える以上、102 条 b 項は適用されないというふうに、反対意見ですけれども判断されているということです。

最後に一言だけ申し上げますと、結局 102 条 b 項の考え方が両者で違っているという意味で、CAFC と連邦最高裁の反対意見と Oracle 側が言っている 102 条 b 項というのは、飽くまでも、アイデアと表現の 2 分論を立法化しただけにすぎない、だから表現に幾つか選択肢があれば、その段階で 102 条 b 項はクリアされる、という立場です。それに対して連邦地裁であるとか Google の主張としては、102 条 b 項というのは特殊な抗弁として、単にアイデア表現の 2 分論を立法化しただけではなくて、コンピュータプログラムという特殊性を踏まえて、method of operation というものに機能性がある場合には、機能部分をどう表現するかが幾つもあり得たとしても、飽くまでもそれは機能の実現にすぎないのであるから、そういった場合には著作権の保護を及ぼさないという趣旨を 102 条 b 項として入れたのだという主張、ということになりまして、102 条 b 項の解釈の仕方がこの論点の判断の分かれ目になっているということです。

3　第 3 回委員会−「Java API」についての理解／Google v. Oracle 事件米国連邦最高裁判決における API の著作物性

(1) サマリー

前記の通り、第 2 回委員会では Google v. Oracle 事件の概要に関する質疑等を行う時間を持つことができなかったことから、第 3 回委員会（2023 年 1 月 16 日開催）は、第 2 回委員会における報告内容を踏まえつつ、改めて、Google v. Oracle 事件米国連邦最高裁判所判決における「Java API」の著作物性に関する議論を参照しながら、コンピュータプログラムの著作物性、創作性について検討する回となった。

開会後、まず、奥邨委員による「Google v. Oracle 事件最高裁判決」と題する資料のうち前回第 2 回委員会では時間の都合上説明が割愛された「Java API」に関して奥邨委員の理解するところについて、前回第 2 回委員会で技術者のゲスト講師・野山孝太郎氏からあった技術的側面からの説明も踏まえつつ、報告がなされた。

その後、平嶋委員から、「Google v. Oracle 事件連邦最高裁判決における API の著作物性について―議論のための素材提供」と題する資料に基づき、Google v. Oracle 事件米国連邦最高裁判所判決が「Java API」の著作物性に関する判断を回避したことに対する同裁判所 Thomas 判事による反対意見の概要や学界における評価等について、また、日本法に置き換えた場合の著作物性についての考え方について、報告が行われた。

以上の報告の後、「Java API」の著作物性に関して、委員間で質疑応答及び討議が行われた。

(2) ディスカッション

ア　Java の仕組み等

(ｱ) 報告「Google v. Oracle 事件最高裁判決−Java の仕組み等」（奥邨委員）（第 3 回 資料 1「Google v. Oracle 事件最高裁判決」）

【奥邨】　前回教えていただいところとも重なるところで少しお話をさせていただいた方が良いのかなということで、省略したところを少しお話しさせていただきます。

a　java の仕組み

（スライド 38）これは前回御説明いただいたところを再度整理したようなものになりますけれども、今回問題になっている Java は、言語仕様、コンパイラ、実行環境（仮想マシーン＋API プログラム）から成ります。

仮想マシーンというのは Java 専用のバイトコードを機械語に変換するインタープリタであって、これのおかげで Java は Java のソースコードを 1 つ書けば、どういう OS の上でもプログラムが走るということになっています。

毎回毎回コンパイルすると時間がかかりますので、Java バイトコードに 1 回変換をします。

基本的な Java の命令・機能は標準クラスライブラリとして用意されていて、それをユーザーがプログラムから呼び出す仕組みがここで言われている API で、この仕組みのおかげで、基本的な機能についてユーザーはいちいちソースコードで記述しなくて済むことになっている。もちろん記述したければ一から全部記述しても良い訳なのですが、そうしなくても済むという仕組みになっています。

それで、一般的には、先ほど申し上げたように、Java については、1 つソースコードを書けば、それを Java コンパイラでコンパイルして Java バイトコードにしたプログラムをどの Java 仮想マシーンにかけても機能するということに、本来はなります。

b　事実関係

（スライド 39）ただ、Android の場合は、この仮想マシーンがそもそも違うということがあって、したがって、Java で通常ソースコードで書いたものはそっくりそのまま Android 用の仮想マシーン上では動きません。極端に言えば逆もまたしかりということになっているので、Java という言語の文法でソースコードを書くというところは一緒なのですけれども、本来、ソースコード

を文法に従って書けばどの OS 上でも動くのだけれども、今回は仮想マシーンが違っていて、また、API も別のものになっているために、通常の Java 用に作ったプログラムは Android 上で動かないし、逆もしかりということになっています。

では、なぜわざわざ Java の文法を使ったプログラムで Android を実現したかというと、Java のプログラムを書ける人は世の中にたくさんいて、そのプログラマーの能力をいかしたい。全く同じプログラムを右から左に流せない。サーバー用などとして動いている Java のプログラムをそっくりそのままスマートフォンで動かすというのは無理もあります。多分今は大分できると思いますけれども、Android が出始めた頃は全くスマートフォンの方が非力でしたし、メモリも小さかったので現実的でなかったというのもあります。また、サーバー用とモバイル用であれば、例えば電源の管理とか通信用のいろいろな機能など、Android には必要だけれどもサーバーには不要、逆にサーバーには必要だけれども Android には不要というものもいっぱいありますので、そっくりそのまま移す必要はない。したがって、サーバーや通常のパソコン用に作っている Java のプログラムが Android で動かなくても別にそれは構いません。しかし、Java のソースコードが書ける人が Android の世界に来てくれれば、Android を一から立ち上げるのですけれども、これからたくさんプログラムが広がることを期待した訳です。

（スライド 40）それで、Android の API についてです。

Java の API というのは全部で 166 パッケージがあるのですが、そのうちの 37 パッケージは丸々宣言コードの部分を利用した。一方で、Android 独自に 131 のパッケージを用意した。少しだけ数字が違っているかもしれませんが、そういう形になります。

（スライド 41）これはブロックダイヤグラムで、最高裁の判決に付いていたものですが、アプリの方で“java.lang.Math.max”と書くと、Sun の Java の API の“package java.lang”の関係するところが呼び出されて、実体である Implementing コード、実装コードが動く仕組みになっているという説明がされている訳です。

ただ、この絵というのは、ブロックダイヤグラムだから良いのでしょうけれども、非常に分かりづらいと思っております。というのは、これですと Declaration Code と Implementing Code が、極端に言うと全く別に存在する、別のモジュールなり別のプログラムとして存在するかのようにしか見えない訳ですけれども、実際は、前回もお示しいただいたものを私なりに色付けしていますけれども（スライド 42、43）、API のプログラム全体というのはこういう形でずらーっとつながっている。この中の黄色い部分が宣言コードで、青い部分が実装コードということになる。

したがって、この最高裁の絵（スライド 41）で言えば、Declaration Code と Implementing Code が別物みたいに書かかれていますが、そうではなくて、つながって存在する訳ですね。それで、“java.lang.Math.max”と言うと、長いプログラムの中の対応するところから動き出しますよということを言っていることになる訳ですね。java.lang.Math.max というふうに言われて、(4, 10)というのを渡すということになると、長いプログラムの中の java.lang.Math.max が特定されて、そこから後の実体のところが動く。それで処理は終わって、アプリプログラムの方に処理が帰っていくという仕組みになっているということかと思います。

それで、API という言葉自体が非常に誤解を招くような状況にあるような気がいたします。

Google v. Oracle 事件に関する議論では、「Java API」という言葉が何を指しているかというのが、前回もお話がありましたが、非常にルーズに使われているというところもあって、論者によって、特に判決だけ見ている法律家と技術の方との間でも必ずしも一致していない。

そしてまた、Java の世界での API の捉え方と一般的な API の捉え方もごちゃごちゃになってしまっているところがあるのではないか（スライド 44）。

Java API というのは、Java プラットフォームが用意する標準クラス・メソッド、基本命令とその機能を利用する仕組みのことであって、問題は、この仕組みの中には実は 3 つのものが混じってしまっていて、正に命令とその機能の呼出し方法という部分、呼び出される機能、命令そのもの、そして呼び出された命令を実現する実体としてのプログラムという 3 つが、「Java API」とルーズに言われたときにイメージされてしまっている。

判決が Java API の「宣言コード」と名付けているのは、この③の「標準クラス・メソッド（≒命令とその機能）を実現するプログラム」の中のクラス・メソッドの宣言部分ということであって、先ほどお見せしたこの黄色い部分（スライド 42、43）だけを指しているということになるのではないかと思います。

この③のプログラム中の黄色いところは、実は①や②の影響を強く受ける部分なのですね。そこに著作物性があるか、そして、そこを利用するのはフェアユースかということが議論になっただけであって、①や②が著作物かであるとか、これに著作権が及ぶかとか、これの利用がフェアユースかというふうなことが議論された訳ではないということになります。飽くまでも、この黄色いところ（スライド 42、43）だけが著作物か、そして、それは仮に丸ごと利用したとしてフェアユースになるかどうかということが議論されました。

そして、この黄色いところというのは、極端に言うと、自由に書けるものではなくて、この①の「標準クラス・メソッド（≒命令とその機能）の呼出し方法」や②の「呼び出される標準クラス・メソッド（≒命令とその機能）そのもの」（スライド 44）によって大きな影響を受けて存在するということになるのかなと。

c　今回の問題を喩えると・・・

（スライド 49）今回の問題に例えるということで、私なりの理解を少し示しました。

野球の公認規則を見ますと、「1.00 試合の目的」、「2.00 競技場」、これが例の日本ハムのスタジアムが小さいとか大きいとかで問題になったところで、競技場の設定などがある訳です。試合の進行などが野球の公認規則で決まっている訳ですね。全部ルールが書いてある訳ですから、法律と同じようなものですね。

X さんが、これの逐条解説である「野球規則逐条解説 X」というのを作るとします。このときに、「1.00 試合の目的」の 1 行目に、「本章は試合の目的について述べる。」と言って、その後は逐条解説としての解説文章をその人なりに書いていく。「2.00 競技場」で「本章は、1）競技場の設定、2）本塁、3）塁・・・から構成される」と、本の各章に何を書いているかということをヘッディングする簡単な 2、3 行があって、その後、逐条解説がどんどん書かれていく。

今度、Y さんが、「野球規則逐条解説 Y」というのを書くとします。Y さんの項目立ては野球規則の逐条解説ですから、X さんのものとそっくりになる。そこだけではなくて、Y さんは、「本章は試合の目的について述べる。」とか、「2.00 競技場」で「本章は、1）競技場の設定、2）本塁、3）塁・・・から構成される」というところも丸写ししてしまった。しかし、解説文章自体は全部自分で書いた。そういうのが実は今回の話だったのではないか。

正に、「本章は試合の目的について述べる。」とか、「1）競技場の設定、2）本塁、3）塁・・・から構成される」と書いてあるようなところが宣言コードであって、逐条解説の解説文章本体自体は実装コードと言われるところで、ここは、Google はコピーをしていない訳です。

（スライド 50）「X は公認規則の項目、その体系・見出しをそっくりそのまま利用した逐条解説を執筆、各章・各節の冒頭には、その章などの構成内容を簡潔に呼べる行が存在する。その行のあと詳しい解説文章が続く。Y は X の逐条解説の項目・体系・見出しと各章・各節、冒頭の構成内容を簡潔に述べる行はそのまま複製。具体的な解説文章自体は Y が独自に著作」。野球規則の項目・体系・見出しが Java 言語の命令語や文法に相当し、各章の冒頭の簡単な文章というのが今回で言えば宣言コードに相当し、詳しい解説文章は実装コードに相当するのではないかと。

こういう状況を前提に考えたときに、各章の冒頭の簡単な文章の 1 行 1 行やこれらの行の組合せに果たして本当に著作権があるのだろうかという話です。

それから、もう 1 つは、これを仮に著作物としたとして、丸々使ったところで、それを著作権侵害とすべきなのかということ。米国流にいえばフェアユースなのかどうか。日本法で言えばどの規定か、私は引用でも良いのではないかと思っていますけれども、そういうことが言えないのかと、そういうことなのだろうなと思っているということです。

前回お話しできなかった、省略させていただいた部分は以上になります。

イ　API の著作物性

(ｱ) 報告「Google v. Oracle 事件連邦最高裁判決における API の著作物性について―議論のための素材提供」（平嶋委員）（第 3 回 資料 2「Google v. Oracle 事件連邦最高裁

判決における API の著作物性について―議論のための素材提供」）

【平嶋】　では、別冊パテントに書きました資料[8]をベースに御説明させていただきます。

a　Google v. Oracle 事件連邦最高裁判決―Thomas 判事による反対意見への注目

Google v. Oracle のうち、著作物性のところについて特に議論をということでありますので、Google v. Oracle の最高裁の Thomas 判事が反対意見を出されているということについては奥邨先生の方から既に御説明があったところだと思いますけれども、着目する点としてはこれが一つあると思います。

この反対意見は、著作物性について多数意見が判断を示さないでフェアユースにより非侵害というふうに判断した点を批判しているということになる訳なのですけれども、これについては、宣言コード部分の著作物性のところについて多数意見の判断についての批判があって、ここに書きましたように（配布資料 1 ページ）、要するに、宣言コードというのは多数意見ではロジカルに考えると機能的であるということ、先ほどの奥邨先生の御説明にもありましたように、機能的な部分というのは大量にあるということだと思うのですけれども、著作権法 102 条（b）の操作方法に当たるということなので、著作物性なしという考え方が念頭に置かれていることが考え得るけれども、それは基本的に間違っているのではないかという、著作物性の考え方についての批判が示されています。本来であれば、この判決では著作物性についての検討は当然なされるべきであったのではないかというのが 1 つの批判の骨子になっている訳です。多数意見がそもそも判断を示さなかったのがおかしいのではないかというのが Thomas 判事の反対の柱の 1 つになっている訳ですけれども、むしろそこは積極的に評価されるのではないかというのが考えとしてあるようです。

機能的という意味では、いわゆる実装コード、実際にバイナリに変換されるベースになるようなコードも宣言コードも変わらないではないかという話と、それから宣言コードを用いるというアイデアの部分はそもそも著作権法の保護の対象にはならないけれども、宣言コードも記述しているという意味では変わらないではないかということで、そこの区別をするということで実質的にコンピュータープログラムとしての表現である部分を、変わらないところについて分けて、あるものは著作物性について積極的に評価しない、片方は積極的に評価するという区分けをするということは論理的によろしくないのではないかというようなところが批判の骨子としてあった訳です。

b　学説における議論と私見-著作物性の評価に関連して

Google v. Oracle は Java の API の著作物性の評価を検討するチャンスだったということがあって、そこは当然期待されていました。一般的には、そもそもそれは否定すべきではないかという考えが、確かに学説などでは、Menell 先生などは地裁あたりからもかなりいろいろ議論されておられて、大きな論文としてはこういったもの（例えば、Peter S. Menell,“RISE OF THE API COPYRIGHT DEAD?: AN UPDATED EPITAPH FOR COPYRIGHT PROTECTION OF NETWORK AND FUNCTIONAL FEATURES OF COMPUTER SOFTWARE”, 31 Harv. J. L. & Tech 305 (2018)）が書かれている訳なのですけれども、こういう考え方の基本というのは、確かに古くから、いわゆる Computer Associates Int’l, Inc. v. Altai, Inc.判決や Sega Enterprises Ltd. v. Accolade, Inc.,判決以来、インターフェースみたいなものの話とか、それから compatibility とか interoperability といったものの確保のために、著作物に該当する可能性があるものであっても、互換性の確保ということを考えたら、一律に著作権法の保護の対象から除外するという発想が、アメリカにおいても下級審で出ていた。一番典型的なのが Lotus Development Corp. v. Borland International, Inc.,辺りで、これは階層構造の話ではある訳ですけれども、こういった議論がそもそも積み重ねられてきたという話です。

Google v. Oracle 事件も、先ほどもお話がありましたが、そもそも宣言コードというのは先ほどの 102 条（b）の操作方法に該当するのではないかという主張もされてきたというところだった訳ですね。ここについては、要するに CAFC、控訴審のレベルで一旦肯定的な判断がされていたところではあるのですけれども、学説はそれに対して割と批判的な考え方をずっと取っていて、Menell 先生はじめ、Pamela Samuelson 先生など、API の著作物性というのは、確かに一般論としては著作物

[8] 平嶋竜太「情報技術イノヴェーション促進と著作権エンフォースメントの調整法理としての fair use」別冊パテント 75 巻 25 号（2022 年、日本弁理士会）171 頁。

になり得るとしても、それはインターオペラビリティの互換性確保という観点からも否定されるべきではないかという考え方は、学説などで強くあったというところであります。

そういう意味では、今回の Google v. Oracle の最高裁判決というのは、そこの部分はある意味肩透かし的に判断しないで、仮に著作物であるという前提を取ったとしてもそもそもフェアユースなのだからと、そちらの方で結論を出してしまったというところではありますので、その点は本来著作物性を正面から判断するべきではなかったかという批判的な議論はある訳ですね。

また、学説で興味深いものとして、ロジカルというよりは、最高裁の判断というのを、そもそもどういう裏があってこんな判断に持っていったのかというようなことに関わる話だと思う訳なのですが、コロンビア大学の若手の、著作権法とか、どちらかというとコモン・ロー、英米法などをかなり研究されているという意味で、知的財産法のプロパーの研究者ではない方かもしれませんけれども、Shyamkrishna Balganesh 先生がこの判決についていろいろ検討されておられて、やはりフェアユースの適用という意味では、かなり far-reaching ではないか、行き過ぎな部分があるのではないかということで、フェアユースの考え方として、これが良い方向に行くのか悪い方向に行くかということはともかく、これによって非常に新しい方向性に出てきたのではないか、とされています。

それで、なぜ著作物性の判断に最高裁が踏み込まなかったかということについて、この方の考えによると、要するに著作物性の評価について、特にそれを否定的な見解をするという立場を取ってしまうと、やはりプログラムの著作物性ということで、先ほどの Thomas 反対意見でも示されていますように、プログラムとしては同じなのだけれども、ある要素の部分については著作物性を否定すると、そうでないもの、いわゆる実装コードのようなものについては否定的な考え方を取らないということで、プログラムのうち著作物性を有すると評価されるものと評価されないものというのが結果的に分かれてしまうということで、そうすると、プログラム全体についての著作物性の評価が非常に否定的なものになるのではないかという考え方、懸念が、最高裁の判事の中、oral argument の中で出てきたということで、余りプログラムの著作物性について踏み込んで、あるものについては著作物としての保護を肯定し、あるものについては否定するという考え方については、割とネガティブな考え方が出てきたということです。

ただ、この場合に権利行使を認めるのかということについては、否定的な考え方が非常に強い。むしろこれは IT の方の、著作物かどうかということに関わる業界の方から否定的な考え方も強かったのではないかということで、両者を調和させるという、ある意味、落としどころようなところで、フェアユースを使って、結論としては否定という方向に持っていったのではないか、という分析が示されております。

そういう意味では、結論的に、著作物性の正面突破の判断をするというのが本筋として確かに期待されていたところが、なぜかフェアユースのところで片を付けたというところに行き着いた背景事情ではないかという説明がされていて、それはそれなりに説明として成り立ち得るものなのかなというふうに考えております。

API の話について、せん越ながら「私見」ということなのですが、これについては、そもそも、著作物性の評価以前の問題として、前回の奥邨先生のお話と富士通の方に御説明いただいたように、そもそも Java については Sun という会社が開発をしていて、Sun を Oracle が全部飲み込んで、Oracle の著作物になったといういきさつがあります。

すると、Sun の段階で、Java がどういう形で開発がされてきたかという経緯も実は引きずっているわけでして、以前御説明がありましたようにいろいろな系統で Java が開発されてきたわけですけれども、コミュニティ、つまり Sun 本体の外部で開発を促進してきて、個別の仕様についてはコミュニティが参画することで実質的にいろいろな傍流が作られてきました。その成果物のうち、ここを標準化しましょうという形でオーソライズし取りまとめてきてできてきたところがあります。そういう意味では、Sun が完全に独立して自社開発を行っていない、オープンソースソフトウェアとは少し性質が違いますが、コミュニティの参画が必須でできあがってきた経緯があります。実は、Oracle が Java を引き継いだ後も、Oracle 自体のオープンソースライセンスでの Java の提供も行っていて、API もその流れの中でアップデートされてきた経緯があるようです。

すると、こういう一連の形成の特殊性は無視できないところがあるようで、元々ある程度解放されたところで開発がされてきたという状況の下で考えると、Java API の著作物性があるかないかで、仮に肯定的という前提で考えても、それを Oracle が引き継いだからその著作権の行使を任意にできるかについては、かなり違和感があるのではないかということで、結論的にはフェアユースを使う

ことで、著作物性の話に踏み込まなくてもそんなに違和感がないとも考えられるわけです。

ところが、これとは別の問題として、そもそも、フェアユースの従来の考え方と今回 Google v. Oracle で示されたフェアユースの考え方がうまく整合しているかは、また別の問題だろうと言えると思います。

Java API の問題とは別に、仮に同じような Java API みたいなものが開発されたとしても、例えば完全にプロプライエタリに一つの企業グループで開発されて、ライセンスとして提供されているプログラムであったとすると、同じようなロジックでフェアユースを使って、第三者がそれを利用することを肯定できるのかという問題は、妥当性というところでは結論が変わってくるのかな、と考えられるところです。すると、著作物性の問題に正面から踏み込んで判断せざるを得なくなるのではないかと思います。

そのように考えますと、この事案の特殊性から考えると、フェアユースを使って結論を出したことは、結果から見るとそんなに違和感がないかもしれないけれども、著作物性のところをいろいろ違う前提で考えると、検討せざるを得ないところはどうしても出てくるのであろうと考えられます。

Java API の複製利用については､先ほど奥邨先生からもバリアフリー化というお話がありました。Google Android の開発用に取り込んだということですが、Java は通常のパソコン等のプラットフォームを前提として開発されているところがあり、そもそも大分違う前提で作られています。けれども、Java API を使ったのは、できるだけ広くいろいろな開発者が参入して Android のアプリを開発できるようにしようという取り込みを図ったところがあります。そこを考えると、実は、インターオペラビリティの問題は、先ほどの従来の学説の中で議論されていた話と、Java API におけるインターオペラビリティの問題や互換性の話が同じような性質の話かというと、大分違うのではないかと考えられます。

先ほどありましたように、Sun は元々Java の開発をかなりコミュニティに任せていたところがあったのですが、では Sun はどこで収益を得ていたかというと、コミュニティの開発のうち一部を Java のオフィシャルなものに適合していると認証するビジネスを行っており、そこが元々 Sun の Java のビジネスの核だったようです。いわゆるソフトウェアのライセンスのビジネスそのものよりも、認証の方がメインにあって、第三者にディストリビューションを積極的に任せるところがメインだったことを見ても大分違っています。

Google には Android のアプリを使う開発者を積極的に取り込むことがメリットになりますが、Oracle の方は皆が Java を使ってくれたおかげで何かプラスの要素が出てくるかというと､余り影響を受けない。インターオペラビリティといっても、片面的というのか、Oracle 側にとっては Java を使われることによって良い方向にいくことは余りない。どちらかというと、Android アプリの開発者の取り込みという､Google 側にベネフィットが偏っている部分があります。インターオペラビリティといっても、かなり違う環境の下での互換性、飽くまで開発者の移行をしやすくするところが基本にあると考えますと、かなり一方向的です。

従来のインターオペラビリティの話では、win-win のモデルを前提にした議論が念頭に置かれていることが多いと考えられることからすると、それをそのまま適用できるかどうか、ややちゅうちょが出てきます。Samuelson 先生の論文などは片面的ということにはこだわらない立場をとっておられるので、そう考えればこの問題はそれ程気にならないかと思いますが、そこはちょっとどうなのか、というのは出てくるところです。

後は、宣言コードをそもそもアイデアと同じだと考える立場ももちろんあり得ると思います。奥邨先生の御説明にもありましたように、宣言コードという領域は名称の記述部分にすぎないというところもありますので、アイデアと同質だと考えれば、そもそも著作権法の保護の対象に入る領域ではないとの切り分け方もあるかもしれません。ただ、そうは言っても、飽くまで記述だということで、どのような書き方をすることも可能ではあります。書く内容はある程度限られるが、いろいろな書き方が不可能ではない領域とも言え、その部分をアプリオリに著作物性がないと言い切ってしまって良いのか、疑念も残ると考えています。

c　日本法に置き換えた著作物性の検討

次に、仮に同じような事案、というのは起こりにくいと思いますが、日本法の下でこういう紛争が起きた場合の著作物性をどう考えるか、検討しました。細かいところは元の論文にいろいろと書きました。その後考え直した方が良いところもあるのかもしれませんが、取りあえずそれに沿った

形で御説明させていただきます。
まず、先ほど言った、Java API 全体の著作物性について、著作権法でいうところのプログラムの「規約」に当たると考えることもできるかどうかという議論です。ここで言う Java API は広く一つのまとまりとしてのパッケージ、パッケージの中に含まれているクラス、クラスの下にあるメソッド、そうした記述表現の全部の塊を全てトータルで捉えたときに、それを規約と言えるかどうか、という議論になります。
「規約」については、著作権法上「特定のプログラムにおける前号のプログラム言語の用法についての特別の約束」と定義されていますので（著作権法第 10 条第 3 項第 2 号）、規約と考えることも可能かもしれません。いわゆるインターフェース、中山先生はインターフェースと言っていますが、どこまでをインターフェースの概念として含めた前提の議論なのかというところがありますが、一つの考えとしてインターフェースと見れば、API という一つまとまりを規約と捉えることもあり得るかと思います。
ただ、これは Java API のうちどこの部分を規約として取り出すかというところがあります。先ほどあったように、いろいろなプログラムを動作させるための操作の塊をパッケージに分けて、更にそれを個別のクラスに分けて、個々のクラスを構成する要素をメソッドというふうに個別に全部定義して具体的な言語で書くという形でヒエラルキーのような構造を作る構成の部分については、用法のようなものと考えることは当然できるだろうし、Java でプログラムを組むとなったら、API という要素をそれぞれ利用してプログラムを組むという、一つの決まり言葉みたいなものとして捉えれば、ここでいうところの構成の部分を規約として捉えることはできるかと思います。
では、個別の宣言コードの部分を規約とまで言えるか。これをアイデアそのものと言ってしまえば身も蓋もないのですが、個々の表現を規約とまで言えるかは、若干検討の余地があるところかと思います。
先ほど申し上げた、アイデアそのものという考え方は、ここで下線を引きましたように（資料 6 ページ中ほど）、構成そのものも、プログラムの言語体系についてのアイデアそのものだと考えれば、アイデアと捉えることもできますが、個々の宣言コードまでを規約とまで言えるのかというところについては、やはりもう少し検討の余地があるのではないかと考えられます。（スポーツやゲームの）ルールと同質と言えるかはちゅうちょがあると個人的には思っています。
プログラムの著作物になるか、を考える必要もあります。別の考え方をとると、実は宣言コードは、プログラムのレベルとして、電子計算機に対する操作の指令という捉え方に当たるものではないのではないかという捉え方も可能かもしれないということで、検討の余地があると考えました。
例えば一つの例としては、昔日本で IBF ファイル事件がありました。Windows になる前の MS-DOS の時代、GUI まがいの、個々のファイルを呼び出すときにその引継ぎをする、ある意味言語的なファイルがありました。それを著作権法でいうところのプログラムと言って良いのかどうかが争われた事案でした。Java API を、本体となる実装コードを呼び出す、間をつなげるものと捉え、ある意味 IBF ファイル的なものと考えれば、そもそもデータに近いもので、いわゆるハードウェアに対する指令ではないという捉え方もできるかもしれません。
ところがこれは、検討の余地はあるのですが、Java API といえどもやはり宣言コードが変換されて実装のレベルに移行していくと考えますと、完全にデータファイルのような構成と言えるかどうか。変換をどう捉えるかによるわけですが、最終的には当然、実装コードに移行するところをみると、やはりただのデータ、著作権法でいうプログラムに当たらないとまでは評価しにくいのかなと思います。IBF ファイルのような事件に合わせて、ただのデータだと言って著作物性を切り捨てるのも厳しいのかなと考えられるわけです。
そう考えますと、最終的には、API を構成する個々の宣言コードそのものの著作物性について類型的に否定することには慎重であるべきだと思っております。実際に見ますと、Java を構成する、本件でも問題になったように様々なパッケージがあり、パッケージの中に更にいろいろなクラスが含まれて、クラスにそれぞれの名称が付いていて、クラスの中にまた更にいろいろなメソッドが置かれていて、それぞれ記述がされているところがありますので、そうした表記部分の表現の書き方をいかに決めるか、規約としてはある程度制約を受けるといいますか、ある操作をする以上、例えばセキュリティに関するような処理方法だと、“java.security”うんぬんといろいろものを特定したり、インターフェースや I/O といったところで、言語的には I/O とかセキュリティとかチャンネルといったことを“java”の後ろにくっつけるというように、書き方がある程度決まってはくるので

すが、その書き方が、全く表記の自由がないとまで言い切れるかと言うと、そこまで言えないのではないかというところがあります。かなりのものについては、そもそも確かに自由が限りなく低いということで、ありふれた表記で著作物性がないと言い切れるかもしれませんが、では類型的に全て否定することには、少し慎重である必要もあると考えております。

場合によって、ものによっては、著作物性、創作性は極めて低いが、著作物として評価し得るものも、場合によっては混在するという考え方もとることができると考えられます。すると、仮に著作物性の低いものがあるとすると、その部分の表記を宣言コードとして丸々Google が Android のベースに取り込んでいるとなると、その部分の権利行使を完全に否定するのも、逆に、著作権があるという前提でどうするか、ということを考える必要性が出てくると考えられます。

そうしますと、日本法の下ですと、フェアユース法理に相当するものは、当然明文による規定はありませんが、現行の規定でいうと、30 条の 4 のようなものが使えるかどうかということも一応視野に入れて検討する必要があると考えられるわけです。

実際、本件で問題になった Java API の表記を見ると、確かに“java.lang”とか“java.security”とか、少し創作性があるとすれば、“javax”にいろいろ付いているようなパッケージ名称があって、更にその下にいろいろな名称がつながっているところを見ると、そんなに創作性が高いとはいえないものがほとんどだとは思うのですが、ただ、全否定できるかというと、やはり個別に評価せざるを得ないのではないかと考えています。場合によっては著作物性ありというものがあってもおかしくはないと考えられるものが、日本の著作権法を前提として考えると出てくるのではないかと思います。

最後は、プログラムの表記と違う部分ではあるのですが、例えばこんな古い裁判例がありまして（古文単語語呂合わせ事件、資料 8 ページ）、ちょうど今受験シーズンですが、古語の語呂合わせを覚えるのに、略語で覚えやすい表記にする表現をいかに作るかという目的で作ったものに著作物性があるかないか、評価された事案があります。

Java のプログラマーがどうやって仕事されているのか私も分かりませんが、恐らく Java API は、プログラムを組むときに、宣言コードを覚えていて、なじむような表記ということで、使われているのかなと思います。そう考えると、この古語の語呂合わせ表記のようなものに機能的には近いところがあって、一部抜粋しました。

「朝めざましに驚くばかり」、「ゆううつな井伏氏」など、どれも見ようによっては著作物性はないと判断することも可能ですが、裁判例では、あるものについては著作物性がある創作的な表現だと肯定する。例えば「志賀直哉もガーナチョコレートを食べたい」は著作物性を肯定して、「バーヤも行きたい大学へ」という表記はありふれていると言っています。個人的にはどちらが著作物性があるのかないのか、違いが合理的に判然としません。

そういう意味では、著作物性の創作的表現の水準というのはそんなに高いものが多いと思えませんが、その中で若干違いがあると評価されているものもあり、日本の裁判例ではこのように個別に検討していることから見ると、Java API も、表記的に普通はこういう表記をするだろうというものがたくさんあって、それがかなりの部分を占めるのだろうと思いますが、あるものは違う表記のやり方もあるが、あえてこういう表記をするというものも全くないとは言えないと考えられます。そう考えますと、やはり完全に著作物性を類型的に全部否定することもなかなか難しい。もちろん、技術的なアイデアだと全部割り振ってしまって、著作物性を類型的に否定するという考え方もあり得るかもしれませんが、著作権法の普通の考え方をプログラムの表現について当てはめて考えますと、ひょっとすると著作物性ありというものも出てきてもおかしくないと考えた次第です。

では、その先どうなるかということになると、権利行使の問題を全く考えないで良いのかというと、そうでもないと言えるということで、もちろん、必ずしもそういう考え方ではないというものもあり得ると思うのですが、パテント誌に書いた時点では、そのように検討して考えた、という次第です。

先ほどの奥邨先生の設例のように、その表現がある種体系的に書かれているという前提で考えると、見出しの部分は機能的な表現なので、そもそも著作物性として取り得るものはないと捉えてしまうという考え方もあるかもしれませんが、プログラムの場合、そこはいろいろな表記があり得るのかなということで、それをどう定義づけるかという書き方が、可能性としてはそのプログラムを開発した人、最初に API を作る人がどう定義するかという自由度がある程度あるのかなと思ったものですから、こういう結論を導きました。

違う御意見、御異論あると思いますが、議論のたたき台ということで報告させていただきました。以上です。

(イ) 討議

【山神】　平嶋先生、ありがとうございました。

奥邨先生、平嶋先生の御説明はかなり刺激的なもので、委員の方々、多分いろいろと御意見があろうかと思います。

今から質疑応答ということにさせていただきたいと思います。どなたからでも結構ですので、御発言をお願いいたします。

伊藤先生、お願いいたします。

【伊藤】　弁護士の伊藤でございます。御説明ありがとうございました。

私も日本法の下で、この Java API が著作物性を認められる余地が全くない訳ではないのではないかといろいろ理屈を考えてはいたところなのですが、先ほど平嶋先生の御説明の中でも「ものによっては」とありました。ものによっては創作性や著作物性が認められ得るといったときに、例えばこの API でいうと、何千個という API があったわけですが、それがどういう条件を満たすと著作物性が認められ得るか、そういう条件というものはあり得るでしょうか。

例えば、API の名前の付け方とか、あるいは API の中にはパラメーター、引数と呼ばれるものがたくさんありますが、その引数の数とか並び順とか変数の型とか、ひとくちに API といってもいろいろな要素から構成されていますが、どの辺りがこういう場合には、創作性、著作物性に影響し得るのかというところ、御意見などがあればお願いしたいと思っております。

【山神】　平嶋先生、その点いかがでしょうか。

【平嶋】　私も余り確たるものはないのですが、多分いろいろな、表記といいますか、少なくとも引数の順番をどうするとか、そういう部分というのは、恐らくどちらかというと、そもそも Java のプログラムの設計思想みたいなところなので、そこは正にアイデアという話になっていくようには思いますが、端的に言うと、結局宣言コード、先ほど奥邨先生に示していただいたようなプログラムの表記を、java***.***、というふうに、いろいろなパッケージ、クラスを書いて、そのクラスの下にいろいろなメソッドがある。その表記をいろいろな書き方で書いているときに、いわゆる略語というのでしょうか、宣言コードの記述をどういう形で定義づけるというか、表記の部分をどう書くかという、そこの部分で若干多様性が出てくるのかなという気がしています。

個人的に考えている部分というのは、クラスや宣言コードの書き方に多少工夫がしてあるなというものが仮にあれば、そこは、創作的表現といっても創作性は低いと思いますが、著作物性が多少はあるのかな、そういう余地が出てくるのかなというふうに考えております。

ただ、それはものによっても多分、例えば、セキュリティに関するパッケージであれば java.security.interfaces みたいな感じとか、java.security.spec といったように書かれているものもあったりしますが、こういうものだと多分もうほとんど、それはセキュリティに関する命令群みたいなものを取りまとめて書いているということなので、書き方としては security 以外書き方は余りないだろうし、それを更に略語で、security の頭文字三つぐらい取って sec と書いたり、という多様性というのはあるかもしれませんが、そのぐらいのレベルだと余り創作性は出てこないかもしれませんが、割と長い命令みたいなものを普通に考えたら書かないといけないもの、それを割とプログラマーがパッと覚えやすいような形に宣言コードに記述してまとめるみたいな、そういう工夫がされていれば、非常に低いかもしれませんが、創作性が多少は出てくるかなと、そんなふうには今のところ考えているような状況です。

ですので、全体の割合としてはそんなに多くはないのかもしれませんが、では一括して包括的にゼロと言い切ってしまえるかというところは果たしてどうかなと。そういったところで考えている感じです。

【伊藤】　ありがとうございます。それでようやく語呂合わせの事例を先生が出されたことの趣旨がよくわかりました。なるほど、それでラベル付けの仕方に、利用者にとっての、何というか親しみだったり分かりやすさだったり、そういったところでの特徴が出れば、それは創作性を基礎づける可能性もあるよと、そういう御趣旨ですね。

【平嶋】　はい。飽くまでそこの部分ぐらいしか出てこなくて、引数とかそういう、そもそもプログラムの全体の処理に関する部分というのは、それはもうアイデアということでバッサリ切り捨

てても、それはあらゆる API について全て共通する部分だと思うので、そこは著作物性の評価にそもそも入ってこない、一刀両断でも余り問題はないなのかなというふうに思っております。

出てくる違いとしては、いわゆる呼ぶときのネーミングというのでしょうか、ネーミングのところの多様性が若干出てくるかなというくらいで。それもそんなに多くはないだろうけれども、という気はしているという、まだそのぐらいの雑な切り分けですが、そういったところで理解しているかなというところかと思います。

【伊藤】　ありがとうございました。

【山神】　今の点、ほかの方いかがでしょうか。

もしどなたもいらっしゃらなければ少し私も発言させていただきたいと思います。

古文単語の事件なのですけれども、平嶋先生が指摘されたように、これがアウトでこれがセーフというのは、いったいどこで線引きされるのでしょうか、というのは実に悩ましい。判決文は、読んでみると実は古文単語の覚え方の類の書籍というのは世の中にいっぱい出ていまして、原告と被告でそういうものを出していたわけですが、そのほかの出版社のものなども引っ張ってきて、そうすると、これはかなりユニークだと思われていたものが、実はほかのところでも皆さんお使いになっているものがいっぱいあるよといったことがあって、創作性が否定されるなどしています。

Java の方に戻ってくると、多分、メソッドなどの命名規則だと思いますが、それも覚えやすいとかいうところである程度工夫できるのですけれども、例えば、平嶋先生にお伺いしたいのですけれども、かつて当落予想表事件というのがございまして、要するに選挙が近づいてきて、候補者が何人かいて、この人は当選確実とか、この人は少し、落選と当選をさまよっているとか優勢であるとか、いろいろな情勢分析がされていて、その情勢分析のときに◎と○と△と×でやるのか、それともほかの、A、B、C と付けるのか 1、2、3 と付けるのかといろいろあって、結論としては著作物性を肯定した、あの事件については、平嶋先生はどういうふうにお考えになりますか。

【平嶋】　当落選は、プログラムでいうといわゆるスペックに近いというか、結局基本的に設定として、選挙の結果をまとめようというひとつの方向性があったときに、それを表にして、個別の候補者がいて、それを当選か落選か、二択のうちどのぐらいの確率かという話というふうにやっていくと、大体どういうものを要素にしていくかということがある意味決まってくるということかと思うので、そういう意味でいうと、プログラムのどの引数に幾つ取ってという、そちらに近いのかなという気がしています。

だからそこは表記として、目的というか、何かを作ろうという前提で来たときに決まってしまう部分というのがかなりあるのかなというふうに考えてはおりまして、どちらかというと、アイデアに近いという話になってくるのかなというふうに思います。

【山神】　ありがとうございます。多分そういうお考えをする方が多数だと思いますが、実際の事件として逆の結論で、それは政治研究家、政治評論家の方に書かせて、それを週刊誌が提供させて勝手に利用してしまいましたといったものすごく気の毒な例だったということが背景事情にあって、ということでした。ありがとうございました。

片山先生、今お手をお挙げになったようですので、御発言をお願いいたします。

【片山】　ありがとうございます。片山です。平嶋先生、御発表非常に勉強になります。

先ほど、例えば java.lang.math.max といったメソッドコールの、例えば名前付けのやりようによっては、もしかしたら、そういうところも踏まえた著作物性の肯定があり得るかもしれないといったお話だったのかなと思いまして、少し分からないところを確認したいと思います。

一点目は、例えば、この java.lang.math.max、最終的には max、複数の数値の中で一番大きい数字を返すという関数だと思いますが、これが math という中に、なおかつ java.lang というパッケージの中にある、というようなところが、本件の場合は多分 java.lang というパッケージの中に入れないと、一般の Java のプログラムをコーディングする人が同じような呼出しの仕方ができなくなるから、同じように java.lang の中に設けるという必要があったのではないかと思うのですけれども、インターオペラビリティの必要性というところと表裏一体なのかもしれないのですが、java.lang というパッケージ以外のところに分類して作るということは、実際やろうと思えばできたことだとは思うのですが、そうすると一般のプログラムのコードを作る人が、同じ関数だけれども全然違う呼び方をしなければいけなくなるので、それは多分非常に不便だと。そこはインターオペラビリティという観点で必要だということになったということで良いのでしょうか、

ということと、逆に言うと、著作権法上アウトならばインターオペラビリティを認めさせなくても良いのではないかという価値判断ももしかしたらあるのかもしれないなと少し思いました。インターオペラビリティが必要だから著作物性を否定するというようなロジックは、どちらが先なのかなと。

何と言いますか、著作権法上アウトならばインターオペラビリティを認めない、利用できなくなるという言い方なのか、必要だから著作物を否定するのか、少し循環論法的な感覚を抱いたので、その辺について少し御意見をいただければ大変助かります。

【平嶋】　ありがとうございます。

インターオペラビリティの話としては、問題になっていたのは、恐らく、一応著作物として保護される前提の創作的なものが元々あって、それが元々は割とプロプライエタリーに使われていたのだけれど、それがものすごくよく使われるということになると、そこの部分を全くのアウトサイダーがやはり活用したいといったときに、結局そこの部分をいわゆるライセンスのような形を受けることなく使える、解放するという文脈で多分出てきている部分というのが多いのかなと思っています。

多分先生がおっしゃるように、Java API の場合も、どうかわからないが、今、あるクラスに放り込まれていて、このクラスの中に分類されているけれども、しかし技術的には違うクラスに組替えしておくことも可能なのだけれども、それをやると実際すごく使いにくいという話なので、使えるようにしましょうという話になってくると。多分その部分はそもそも著作権法で保護されるかというと、先ほど言った規約の方に結構近いのかなと。「ここの組に置いているというところを保護してくれ。」という議論になってきて、「いや、それはそもそも組の問題だから、著作権法の保護の対象にならない。」と考えてしまうと、そこはそもそも、インターオペラビリティを確保する部分というので、著作権法で保護することが前提にならない部分になってしまうのかなということがあります。

そういう意味では、この場合、インターオペラビリティの部分ということで、今までの議論で言うと、Java API の著作物性といっても、創作性が低いものばかりで、まれに何かあるかもしれないというくらいのレベルで見たときに、ではそれを使えるようにする、といったときに、正面からインターオペラビリティの議論で使えるようにできるかという話になるのかというと、そこは確かに、先生のおっしゃるように、仮に著作物性が認められるという前提のものがあったときにどうかという話になると、インターオペラビリティの従来の議論でうまくいくのかどうかというのは、個人的にはどうかなと思ったところです。そういう意味では、クラス替えというような話というのは、この事案に即してみると、インターオペラビリティの問題として著作権の問題と直接絡むということは余りないのかなという気はしております。

そういう意味では、インターオペラビリティの話と著作権で保護しなくて良いかという話というのは、多分事案によって、場合によっては正面から、元々著作物性に余り疑義がなく、使っている部分を第三者がどうしても使いたい、ゴリ押しで使うというときに従来インターオペラビリティの話というのは割と出てくるのかもしれないというように考えているのですが、いかがでしょうか。すみません、余り質問に対する答えにはなっていないかもしれませんけれども。

【片山】　ありがとうございます。

インターオペラビリティでどこまでがそのまま維持しなければいけなかったところなのかがはっきりわかっていなくて、メソッドの名前と、その中で引数の型、いわゆる関数の呼び方は維持しなければいけないという部分なのかなと思っているのですが、そこはそういう理解で合っているでしょうか。

【平嶋】　そうだと思います。あとは多分、先ほど先生がおっしゃったように、どこのクラスに属させるかみたいなこと、実は違うところに置いても一応使えなくはないけれども、しかしそれを動かしてしまうと従来のプログラマーがここかと思っていたら全然違うところから引っ張ってこなければ駄目だというようなことになってしまうと、それはすごく不便ではないかという話になるのかもしれません。

【野山】　すみません、富士通の野山ですが、少しテクニカルな観点から。

【平嶋】　はい、その辺御教示いただければ。

【野山】　インターオペラビリティという観点だけで語ると、ほとんど変更の余地はないです。java.lang というパッケージが変わった瞬間にエンジニアの方で呼び出す形も変わりますし、メ

ソッド、呼出しの関数名、ここも変わると駄目です。唯一、今仕様などで規定されている呼出しの中で変更してもほとんどエンジニアから影響がないのは、呼び出しているところの変数名、先ほども max の中で、lang 型の a、lang 型の b という二つの引数を渡すような形を書いていましたが、ここの a とか b は c、d だろうと x、y だろうとナンバー1、ナンバー2 だろうと、ここは全然、ほぼ関わりないのですが、それ以外のところは変更されると、エンジニアの観点からいうと、インターオペラビリティはなくなって、それ用にコードを書き直さなければいけないという形です。

すみません、少し口を挟ませていただきました。

【平嶋】　ありがとうございます。そうすると、結局そこを替えてしまったとしたら、少なくとも java としての成り立ちが全然崩れてしまうということですね。言語として。

【野山】　そうですね。少なくとも呼び出すコードなどは書き換える必要が出てくる形です。呼び出し方が変わってくる。

【片山】　非常にわかりやすく、ありがとうございます。

そうしますとそこまでして変えさせるか、いやいやほかの人たちの利益があるのだからそれぐらいは良いだろうか、というようなバランシングということですよね。

【山神】　恐らくそういうことに落ち着いたかと思います。片山先生ありがとうございました。椙山先生がお手をお挙げになっています。大変お待たせいたしました、よろしくお願いいたします。

【椙山】　元々判決を読んだときに、copyrightability という問題の立て方に違和感があったのですけれども、要するに 102 条 b の問題なのでアイデアと表現の区別の話ですよね。

特に、アイデアと表現の区別ということになると、Universal Pictures のような文化的なものの保護範囲の問題と、それから特許との関係で、技術的な保護については慎重でなければいけないという二つの系統の話がありますが、そこで創作性といったときに、著作物一般の創作性ということであれば、例えば日本の判例でいうと、誰がやっても同じにならないほどの工夫があれば良いのではないかというのが普通の基準になっているわけです。

そういう基準からいえば、創作性はあるに決まっているわけです。労力的にいっても工夫からいっても、決まっているわけだけれども、102 条 b のところでやった場合にズバッと切ってしまうとどういう問題があるかというと、やはり技術的なものを保護するということが、著作権法の目的、著作権法の保護にフィットするかどうかという問題があるわけです。

だから平嶋先生の発表にも違和感があったのは、そういう語呂合わせみたいなものの創作性の話と、技術的なものの創作性の話というのは区別して考えないといけないのではないかと。つまり、工夫や努力があったとしても、技術的なものであるが故に著作権の保護は適当でないという分野があって、それに対する配慮があるということだろうと思います。

ただ 102 条 b でやってしまうと、保護するかしないか、オールオアナッシングになってしまうので、そうするともうどんなにたくさんあっても何をまねしても、API に当たるものだったら保護されないということになってしまって、それは最高裁としてはそこまで言うのは嫌だったから、個々の事情でいろいろなことを考えてここはセーフにしておきましょうと。しかし、この手のものはアプリオリにアウトになるわけではないですよと。そういう価値判断を示したのだと思うのですね。そこはごっちゃにしない方が良いかなというふうに思いました。

【山神】　ありがとうございました。今の点なのですけれども、例えば、言語の著作物で、何か文章が書かれているとき、それが機能的な著作物になり得るのかというと、例えば電化製品の取扱説明書とかパソコンの説明書、それから今の古文単語の日本人が聞く語呂合わせも、やはりある程度自由にはできなくて、そういう意味では、技術的という単語が良いかわかりませんけれども、かなり似ているかなと個人的には思いましたが、その辺り、椙山先生はいかがでしょうか。

やはり少し違うものなのだという、おっしゃりたいこともとてもよく分かるのですけれども。

【椙山】　例えば、取扱説明書というような場合だと、言語の中でも機能に束縛されるといった意味で保護範囲が狭くなるというふうに一般化するのだったらまだ分かるのですけれども、日本の判例の、ということになるかもしれませんが、普通の判断基準から言えば、この程度の工夫でも保護されるのか、というものが幾らでもあるわけですよね。だけれども、それに比べれば圧倒的に工夫があって努力もあるのに保護しないというのは、それは別の原理が働いているからということなのだろうというふうに理解しているわけです。

【山神】　ありがとうございます。では上沼委員が手を挙げておられますので、御発言をお願いい

たします。
【上沼】　今のお話と直接関係があるかどうかわかりませんが、奥邨先生と平嶋先生のおかげで、イメージがすごくよくわかりまして、今日は聞いていて良かったなと思っているところです。
　平嶋先生のレジュメの 2 ページ目から 3 ページ目にかけて、最高裁が著作物性のことについて、否定することにすごく抵抗感がある、否定してはいけないのではないかというような価値判断があったのではないか、といったことを書かれていて、御説明もされていたと思うのですが、その辺り、もう少しお聞きしたいなと思っていいます。
　先ほど椙山先生もおっしゃったように、著作物ではないとなると、オンオフで、0 か 100 かというようなことになってしまって、著作物なら保護されるけれど著作物ではないといった瞬間、どこまでいってもコピーし放題というふうになってしまうということについての危機意識なのか、フェアユースだとその辺りが柔軟に処理できるということなのかもしれませんが、その辺りの問題意識のところをもう少し伺いたいなと思って手を挙げました。お願いできればと思います。
【平嶋】　先ほど椙山先生が違和感があるとおっしゃったのですが、私も言語の著作物とこれをいっしょくたで考えているわけではなくて、ただ、今上沼先生がおっしゃっていた質問の答えが実は先ほど椙山先生が先ほどお答えになった話になるのではないかと思います。
　先ほどの論文のところで挙げられた懸念というのが正にそれで、結局、著作物性のところで評価したときに、プログラムの著作物というのは確かに 102 条(b)で、元々プログラムは全部機能的な表現なところがあるわけですが、そこに引っ掛けた場合に、機能的な表現にどこまで上積みの条件を積まないと著作物として保護されないのかという話に踏み込んでいってしまって、ではどういう基準で分けるのかという、非常に厄介なところがあるわけなのですけれど、宣言コードのところは著作物性なし、と言ってしまったときに当然そこを答えざるを得なくなってしまうというか、ではどういうふるいでこの宣言コードは著作物ではないと言えるのかという話に、どうしても踏み込まなければいけなくなるというところがどうしても出てきて、そこの考え方を出すという話になると、恐らく何か基準を最高裁として示さなければいけないというところにまで踏み込まざるを得ないというのは、多分確かにあるのかなと思いまして。
　多分そこのところに踏み込むということになると、これまた非常に面倒くさい話になってきて、では最高裁できっちりクリアな基準を示せるかどうかというところがあったのかなと。そこの部分に踏み込まないと、この事案の結論が出せないのかという話になると、いや待てよと。Java API について、先ほどありましたように、そもそも開発の経緯からしていろいろな要素があって、では結局、元々コミュニティで作っていたという流れのところもあるというような個別の話を考えていくと、結論的にはフェアユースの話の方向で結論を導ける部分があるのではないかというところの方が取り出されたという話になるのかなと。それで恐らくそういう方向に結論として持っていったという話なのではないかと。
　多分椙山先生も先ほどそういう御趣旨でおっしゃったのかと思いますし、先ほどのコロンビア大学の、Balganesh 先生の論文も、そういう趣旨で言っているのかなと。
　oral argument では、一部の判事かもしれませんけれど、判事の中でそういう雰囲気、懸念が出たのかもしれません。多分その話があって、著作物性の話について議論すると、やはりどうしても、機能的な表現であることが前提なのだけれども、ではどういう要素で、プログラムの場合、あるものの表現は著作物性ありで、あるものはなしなのか、という話にどうしても踏み込まないといけないというところかと思うのですね。多分、ではそこに踏み込まないとこの事案の結論は出ないのかどうかというところで、話としてはそこに踏み込まずに終わってしまったということかなとは理解しております。
【上沼】　ありがとうございます。
【平嶋】　椙山先生が先ほどおっしゃった御趣旨はそういうことでよろしいでしょうか。
【椙山】　はい、そういうことです。
【平嶋】　私もそこは、日本法に置き換えたときにどうリライトされるのかというところは定かでないというか、実は、日本の著作権法がプログラムの著作物性に上積み的な解釈をしているかどうかというのは、余り強く印象は持ってはいないのですが、日本の裁判所がプログラム著作物と言語著作物のすみわけで、どのぐらい使い分けしているのかというところについて、もしあれば、また是非コメントなどいただければと思いますが、それで考えると一刀両断が日本でできるのかなというのは少しちゅうちょがあるというように考えている次第です。

【椙山】　プログラムの問題が出てきたときに、中山先生が知財高裁の前身に当たるところで講義されたことがあって、そこで裁判官相手に、プログラムは技術的なものだから保護範囲について慎重でなければいけないとおっしゃったのですね。

それで、その後出てきた判例というのがシステムサイエンス事件の判例と IBF ファイル事件の判例です。基本的に裁判所としてはプログラムの保護については著作権法でするとしても狭く考えなければいけないという、それが少し程度が強過ぎるようになったという気もするのですが、そういう意識がずっとあって、ダブルスタンダードというか、プログラムの場合、ほかの技術的なものや事実に関するものもそういう面があるのですけれども、そういうものについては、著作物の種類によって調整しているのが判例ではないかなというふうに私は考えています。

【平嶋】　ありがとうございます。そうしますと、日本の場合、いろいろ検討したけれど結果的にほとんど著作物性なしになるということも当然あり得るという感じでしょうか。

【椙山】　そうですね。フェアユースみたいなものがないから、日本ではありかなしかになりがちなところがあるのだろうと思うのですけれども、そういう意味で言うと、アメリカみたいにいろいろなことを考慮に入れるのは柔軟で優れたところかなと思います。

【平嶋】　確かに。実は、30 条の 4 の話には今日は触れなかったのですが、30 条の 4 で仮にこれが著作物性ありといった場合に救えるかというと、個人的には結構厳しいかなと。逆に、表現の享受に当たらないと言えるかというのは少し苦しいという気はしていますので。そういう意味では、著作物性ありという網に引っかかるものが出てこない方が望ましい。結果論みたいになってしまうのですが、そうすると少し面倒くさいというのはあるので。

確かに、日本の著作権法では、宣言コードについてはプログラムの著作物の水準からいうとそれほど創作性は高くないということで、著作物性なしという範疇で、プラスアルファというか、基準の評価みたいなところでうまく調整できればそれに越したことはないかなと。

逆に言うと、そのぐらいの範疇のレベルのものかもしれないとは思いますけれども、確かにストレートに言語の話で見ると残ってしまうものが少し出てくるかなという気はしていて、それで今日はああいう例（古文単語語呂合わせ事件）を少し挙げてみた訳なのですけれども。

【山神】　椙山先生、よろしゅうございますか。

【椙山】　はい、結構です。

【山神】　ありがとうございます。そうしましたら、大谷委員、大変お待たせいたしました。どうぞよろしくお願いいたします。

【大谷】　ありがとうございます。大谷です。非常に興味深くお 2 人の発表を伺わせていただきました。

今までの議論と違うのですけれども、今回の判決がフェアユースを使っているというところなのですが、フェアユースを使った場合には、個別の事件についての判断がまずありきで、なかなか一般化しづらい。つまり、同様の事件が生じたときの予見可能性が低いという批判を受けることが多いということで、確か平嶋先生がパテント誌に書かれた、本日御説明いただいていない部分でも、予見可能性の部分についてコメントされていたと記憶しているのですけれども、やはり私自身としては、この事件から、狭いとしても一般化できる要素はないのか、つまり API 全般に同じようなことが言えるかというとなかなか言えないと思うのですけれども、Java API のほかの宣言コードについては当然言えるのだろうということは言って差し支えないと思われますし、類例があるとしたら、どこまでそれが通用するのかといったことについて、感覚的なことでも御教示いただけないかと思っております。

それで、特に Java に限らず、ほかの同様の API に、どこまでどういう条件が整えば、今回のフェアユースの議論と同等の議論ができるようになるのかといったことについての粗々とした見解、認識を共通のものにできたら非常に有り難いなと思っております。

今回はコミュニティで Java の標準化がなされていたという歴史的な経緯、形成過程が判決などにも影響しているかもしれないし、判決の受け止め方にも影響しているという御説明を頂いたところなのですけれども、そういう背景が常に必要なのか、それとも、今回の最高裁の判断では余りコミュニティだからということは言われていないので、私はそこは捨象して判断がなされたものと考えているのですけれども、仮にコミュニティ的な性格があれば同様の判断ができると考えて良いのかどうかについて、平嶋先生と奥邨先生の御意見をいただければと思います。

【山神】　まず、平嶋先生から今の御質問にお答えいただいて、その後、奥邨先生、ディスカッショ

ンの後にお答えいただいても結構です。どうぞよろしくお願いします。
【平嶋】　混乱を招くようなことを言ってしまい申し訳ないのですけれども、Java API の場合のコミュニティの話というのは飽くまでも個人的な受け止め方で、最高裁はフェアユースのところにおいてはその辺りの要素というのは判断要素として全く入れていないので、そういう意味では、最高裁のフェアユースの当てはめ論というところで見ると、やはり特徴的なのは、次回以降この委員会で取り扱えるかもしれませんが、第 2 ファクターからいきなりフェアユースを適用していて、著作物の性質についての判断がいきなり来て、ここでフェアユースの結論のかなりの部分が第 2 ファクターで決まっているような気が個人的にはしなくもありません。
　結局、第 2 ファクターの検討では、実質的には著作物性の判断はしていないのだけれども、本件の場合は、著作物の性質というところでほとんど結論を検討しているように思えて、そうすると、結局、著作物性の話を正面からしはいないのだけれど、フェアユースのところで事実上 Java API のプログラムはどんなものかという検討をして、それがフェアユース全体の話にかなり色濃くほかのファクターにも影響を及ぼしているように思えます。そういう意味では、結局、著作物性の先ほど来の議論に近いことが事実上フェアユースという枠組みの土俵の中に紛れ込んでいるように思えるのですね。
　ですから、そういう意味で考えると、事案の API 全般の話というところでみたときに、個々の API の表現というのがどういうものかという評価がフェアユースの話につながっている部分は非常に大きいように思っています。
　ですから、ひょっとすると、違う API という話になったときも、実は先ほどのコミュニティみたいな話は、飽くまで自分が捉えた上では、Java API には特にそういう経緯があったので、個人的にはそこでフェアユースで結論を導くことにさほど違和感はなかったのですが、判決ではそこは余り考慮されていない気はしていまして、むしろやはり著作物性、copyrightability の議論が事実上フェアユースの中に取り込まれている部分がこの判決はすごく強いなという気もするのですね。
　そういう意味では、その部分、その考え方が一般論としていくのであれば、API の部分も、フェアユースを使うかもしれないけれども、実質、第 2 ファクターみたいなところというのをもしやっていくとすれば、そういう考え方をみて、著作物の質がどのようなものかみたいなところで、例えば Java API 的な性質が強いものですと、同じような話になるのかどうかというところなのですが、ただ、やはりそこがフェアユースの一般論とかなり違うのではという議論があるようにも思いますので、これを最高裁がどこまで広げていくのかというのはどうなのかなという気は、飽くまでこの事例限定的な取扱いでやっているようにも見えなくもないように思うのですが。
　そういう意味では、考えるとしたら、先ほど申し上げたコミュニティみたいな話というのは、恐らく余り影響しないのかなという、飽くまでそれは自分の勝手な邪推なのかなというふうに思っていますけれども。
【山神】　平嶋先生、ありがとうございました。そうしましたら、奥邨先生、よろしくお願いいたします。
【奥邨】　はい、今の箇所で言えば、平嶋先生もおっしゃっておられたように第 2 要素から判断しているというのは通常はない順番と言えると思います。
　ただ、従来、例えば、プレーステーションの事件でしたか、著作物性、著作物がプログラムだった場合の過去のリバースエンジニアリングの事案など、第 2 要素から始まっているものもありますので、そういう点ではやはりプログラム特有の順番と考えることができるのではないかと思っております。
　それで、この判決を引用している判決がどれだけあるかというのを LexisNexis で調べたのですが、ほとんどありません。10 数件しかありません。これだけの最高裁の判決が出たら、普通はフェアユースについて言う判決がどんどんこれを引用するのですよ。しかし、ほぼ引用していないのですね。それで、フェアユースについて言う判決がこれを引用せずに Campbell 事件を引用したりする訳です。
　したがって、この判決が持つ影響力はやはり小さいのだろうなと思わざるを得ない。若しくは、小さいというよりプログラムの事案でしか適用されないという認識だろうと思った方が良いのかもしれない。
　それで、かなり詳しく議論しているものとしては、今最高裁にかかっている Andy Warhol の事

件の控訴裁があって、あれは判断を出した後に（Google v. Oracle 事件連邦最高裁判決が）事案的に出てしまって、しかも被告側か原告側どちらかから求められたので、もう 1 回考え直しても結論は一緒だよという念押しのために出しているだけで、あれをやる必要があったかどうかは少し横に置いてという話だと思うのですよね。そういう点ではやはり特殊性はあると考えるべきだろうと思っています。

それで、より抽象的に考えると、目的として、再実装目的というのはトランスフォーマーティブなのだと言われたということは、コンピュータプログラムの世界、プログラム著作物の世界でじゃ非常に大きいだろうと思いますね。再実装ということが正面から出て、インターオペラビリティとは言っていないのですね。再実装と言っている訳ですので、そこが非常に大きい。

だから、先ほど平嶋先生もおっしゃったように、インターオペラビリティだったらバイラテラル、双方向でないといけないのに、これはワンウェイで良いというのは、再実装だからなのですね。既にあったものを別のプラットフォーム上に移していくことは業界でよくあることだし、あるべきだという判断が裁判所にはあるみたいなのですよね。

そうだとすると、従来のインターオペラビリティなどについて言っていた次元よりも更に踏み込んで、そこにトランスフォーマーティブを認めたということになると、そこは大きいと、私は一般化するならばできるのかなと思っています。

それから、全体の御意見を伺って思った点として 1 点申し上げると、やはり API の話なのか、API の宣言コードの話なのかというのが、一緒になって話をされているような気がいたします。飽くまでも今回議論になったのは、API に基づいて作られた宣言コードの部分の著作物性であって、そうだとすると、先ほど技術の御説明でありましたように、API でこういう順番でこういう名称を使いますというのがもう全部決まっている訳ですから、自由度ゼロなのですよね。

それで、この上に、例えば先ほどの文章（配布資料 42 ページ）ですけれども、“package java.lang”はこれはもう API で“package java.lang”と書くしかない訳でして。しかも、Java の言語としてその後“import”と書くしかない。これは Java の文法で全部決まっている訳ですね。その部分の著作物性が議論された訳ではないということです。そういう決まりに従って書かれたこの黄色い部分（配布資料 42 ページ）が著作物かどうかという議論なのだとすると、私はもうほぼ余地はゼロと言っても良いのではないかと思う次第です。

それで、そうではなくて、Java の API で、この黄色いところ（配布資料 42 ページ）になる前の段階で、野球規則の例え（配布資料 49 ページ）で言えば、野球の公式規則のところで、「競技場」と書くとか、「ゲーム」と言わずに「試合」と言うと決めるというところの議論と、そこで「試合」と書いた、「競技場」と書いたということで、「『試合』の目的」と書かざるを得ないということで、これはもう一対一で対応せざるを得ないので、一方で自由度があったということと、他方に自由度があるかというのは別次元のものとして分けて考えるべきなのではないのか、仮に同じ人が作ったとしても、ものとして別なので、別々に考えていくべきではないかと思いました。

それから、もう 1 点御議論で出た中で言えば、30 条の 4 が難しいのではないかという点については、私も全くそのとおりで、基本的には「表現された思想、感情」を享受目的というふうに言う訳ですけれども、それはプログラムの場合は「機能」と読み替えるというのが文化庁の言い方ですが、そうなると、正にそのまま機能を享受してしまう可能性がありますので、これは難しいのではないかと。

それで、フェアユースというのは日本にないと言うのですけれども、元々よく言われるように、例えばパロディに引用規定が使えるのは、実はアメリカは逆に言うと引用規定がないのですね。フェアユースがあるから引用規定がないということは、フェアユースと引用規定というのは実はある程度類似性がある訳です。しかも、引用の規定を美術品鑑定証書事件で総合考慮説型に読んでしまえばある程度柔軟にも使えるということになると、私は、ほとんどないと思いますけれども、仮にあったとしてもその部分は引用なのだと言ってしまって 32 条で処理するということも十分あり得るのではないかなというふうに思っております。

【山神】　奥邨先生、ありがとうございました。 大谷委員への回答、また、全般へのコメントを頂きました。ほかの委員の方々、いかがでしょうか。

宮下先生、どうぞよろしくお願いいたします。

【宮下】　はい、ありがとうございます。大変勉強になりました。

奥邨先生が御指摘された点、私も少し考えておりまして、相互運用性というと、かつては、リバースエンジニアリングの文脈で申し上げると、垂直的な相互運用性確保のためのリバースエンジニアリングは許容されるけれども、競合品を確保するためのリバースエンジニアリングは必ずしもそうではないというような議論のされ方をしていて、つまり、OS についての市場支配力をアプリケーションにも及ぼすことによって、アプリケーション開発を制約するようなことはよろしくないという発想が根底にあったと思うのですね。

それで、今回の場合は、Oracle が PC 用に作った Java プラットフォームをスマートフォン用のプラットフォームに移植するという意味での潜在的な市場への参入を制限するような側面もあるので、競合品を作ることを許容するための API の利用というような側面も見方によってはあるのだろうという気がしていて、いわゆる相互運用性との関係でどこまで権利が及ぶのかということで、従来リバースエンジニアリングの関係で議論されていたこととは少し観点が違うのではないかという気がしていました。

それで、恐らく裁判所としては、Java というある意味標準的な言語あるいはプラットフォームの技術資産がある訳なのですけれども、それを PC 用ではなくてスマートフォン用にも活用できるように本来すべきなのではないかと。それを有効活用ができないような形で Oracle が権利主張するというのは、フェアユースの観点から適当でないのではないのかという意味で、正に奥邨先生の御指摘のとおり、相互運用性に関する議論を一歩更に進めて、再実装のためなのであれば、API のある意味での利用が許容されるというような考え方の下で、今回のような判例が出たのではと勝手に思ったのですけれども、もしそういう見方に違和感がございましたら、御指摘いただければと思います。

【山神】　奥邨先生、一言いただければと思います。

【奥邨】　はい、私は正にそういうことだろうと思っております。しかも、裁判所が言っている中で、Oracle が得るべきだった利益ということが、結局第 4 要素との関係でも問題になるのですが、裁判所は利益の源泉という言い方をしていて、そもそも著作権の保護対象ではないところだというようなことを言っている訳ですね。

ですから、結局、表現の独占からどこまで保護できるのかということからすると、結局プログラマーがいろいろな経験を積んで身につけた部分を、Oracle が仮に API 全部の著作権を持っていたとして、そもそも API の宣言コードも実装コードも全部持っていたとしても、だからといって、プログラマーが一生懸命身につけたところまで Oracle の著作権が及ぶのか、そんな独占権ではないのではと問いかけているという点で、やはり大きな考え方のポイントだと。

似たようなことは昔、ソニーコンピュータのリバースエンジニアリングの事案でも、そこは著作権が保護するところではないと切っていますし、元々リバースエンジニアリングの議論は、Campbell 事件の判決が出て以降、トランスフォーマーティブがどうこう言う訳ですけれども、先生に申し上げるのも申し訳ないのですが、元々は著作権保護の対象ではないところへアプローチしていくのに、形式的な複製があるから駄目だと言うのは、それはベニスの商人だよね、ということから始まっているところがある訳なので、そうだとすると、やはり裁判所は著作権が保護しないものについて明確に頭にあって、著作権をてことして何とか近づいていこうという今回の Oracle のやり方はおかしいのではないかという価値判断が前提にあった。だから、そういうものを許さない。再実装を認めているのは、飽くまで、著作権が及ばないものの再実装をするために必要な範囲で結果的に付随的にコピーされてもかまわない、という割り切りなのではないかと私は思っております。

【宮下】　ありがとうございます。大変ふに落ちました。

4　第 4 回委員会−Google v. Oracle 事件米国連邦最高裁判所判決におけるフェアユースに関する判断／AWF v. Goldsmith 事件の概要／日本の著作権法のもとでの権利制限規定の適用可能性

(1) サマリー

第 4 回委員会（2023 年 1 月 30 日開催）は、Google v. Oracle 事件米国連邦最高裁判所判決における「Java API」に関する議論を参照しながら、「権利制限」の観点から、米国法におけるフェアユースの考え方を振り返りつつ、我が国著作権法における権利制限規定の適用可能性について検討する回となった。

まずは今回もはじめに奥邨委員から報告が行われた。前回と同じ「Google v. Oracle 事件最高裁判決」と題する資料に基づき、Google v. Oracle 事件米国連邦最高裁判所判決におけるフェアユースに関する判断についての振り返りが行われた後、同判決の今後のフェアユース判断への影響等についての奥邨委員の見解が報告された。

次に、石新委員から、「AWF v. Goldsmith の概要」と題する資料に基づき、Google v. Oracle 事件米国連邦最高裁判所判決後に同裁判所において同判決を踏まえた弁論が行われてフェアユースに関する今後の判断が注目される The Andy Warhol Foundation For The Visual Arts, Inc., v. Lynn Goldsmith, Lynn Goldsmith, Ltd.事件についての報告が行われた。

最後に、伊藤委員から、「―API の法的保護―日本の著作権法のもとでの権利制限規定の適用可能性」と題する資料に基づき、「仮に Java API に著作物性が認められた場合、フェア・ユース規定がない日本の著作権法のもとで、権利制限規定の適用によって Java API を適法に利用することはできるか。」との観点から報告が行われた。

以上の報告の後、「Java API」に著作物性が認められた場合における「権利制限」に関して、委員間で質疑応答及び討議が行われた。

(2) ディスカッション

ア　フェアユース

(ｱ) 報告

a　Google v. Oracle 事件最高裁判決（続）（奥邨委員）（第 3 回 資料 1「Google v. Oracle 事件最高裁判決」差し替え版）

【奥邨】　判決のフェアユースの概要は前回御説明しておりますので、基本的には細かく御紹介するつもりはないのですが、フェアユースについて、何があったのかというところだけもう 1 回おさらいをしておきます。

(a) フェアユースの 4 要素の検討

基本的には 4 要素を検討していくわけなのですけれども、総論のところを今お見せしていますが（スライド 17）、最高裁は実際には第 2 要素から順番に検討していくということを行いました（スライド 19）。この第 2 要素のところ、今回問題になっております API の宣言コードの部分ですけれども、この宣言コードは機能に非常に結びついたものであるということを言って、そのために、仮に著作権が認められるとしても、宣言コードはコンピュータプログラムの中でも著作権保護の核心からは遠い存在だということで、著作物の性質の部分がフェアユースに有利に働くとしています。これが仮にクリエイティブな創造性に富む小説や娯楽などの類いのものであれば、それを利用するのがフェアユースに不利に働くと判断されるのですけれども。これはプログラムの著作物自体がそもそも著作権の保護の中心からそれほど近いところにはないのですが、その中でも更に遠いところにあるというふうに言って、そういう判断を示します。

（スライド 21、22）次に第 1 要素の「著作物の利用の目的」のところですけれども、裁判所はこれについて、Campbell 事件以降、transformative、変容力がある利用かどうかというのを重視してきたわけですけれども、これに照らして、今回も変容力のある利用であったとしています。Google は、基本的には、再実装のために API を利用した。このインターフェースの再実装という

のは、コンピュータプログラムの開発を促進するもので、目的があることから、Google の複製には変容力があると言いました。商業的な利用ですし、Google 自体は無断で使っていることがよく分かっていたわけですけれども、そういったことは変容力がある、transformative な利用の場合は、別段決定的ではないということも言って、変容力があるから第 1 要素についてもフェアユースに有利になるという判断を示しています。

（スライド 23、24）第 3 要素は「利用する部分の量と実質性」を問題にするものです。これは前回もお話が出ましたが、利用しようと思ったところの全部を利用していますけれども、利用したところというのはそもそも Java の API を構成するプログラムのコードのうちの 0.4%にすぎないということを言っています。それから、量と質というのは合わせて考えることになりますので、著作権保護のコアになるような部分であれば、少量の複製でもフェアユースの成立に不利に働くということになるのですけれども、今回は著作権保護の中核からかなり遠い存在のプログラムだと言っていますので、そういうものについては、仮にある程度の量をまとまってコピーをしたとしても、それ自体はフェアユースに有利に働くと言っています。しかも、そこまででしたら中立なのですけれども、今回はインターフェースの再実装を変容力のある利用に必要な範囲で行っているということでありますので、これはフェアユースに有利に働くという判断をしています。

（スライド 25、26）最後は市場に影響を与えるかということなのですけれども、市場に影響を与えるかどうかという点で重要なのは、どういう原因で市場に影響を与えるかということであって、それは、基本的には著作権で保護される利益に対する損害になっているのかと。結果的に Google がやったことによって損害を受けるといっても、飽くまで著作権法の話ですから、著作権法上の利益に対して損害があるのかどうかということを問わなければいけないということですね。

それからフェアユースはもう 1 つ、潜在的な市場に影響があるかどうかということも問題にするのですが、潜在的な市場というのは、考えられる全ての、ありとあらゆる可能性のある市場ということではないと。そんなことを言ってしまうと、全ての場合に損失が考えられてしまいますのでそうではないと。ですから、実質的にそういう可能性があった市場かどうか、現実的な可能性があった市場かどうかということを考えていくと。

そうすると、そもそも Sun、Oracle 側が得られると言っていたのは、実際には著作権そのものに由来する、源泉がある利益ということでもないし、また Sun がそういう部分で実際的な市場を持ち得た可能性は低いということになると、Sun の市場に与える影響、そしてその市場が得られる利益に与える影響というのは極めて概念しづらいということで、これもフェアユースに有利という判断をしたというのが大きな流れとなるわけです。

(b) 検討

（スライド 28、29）これらを踏まえて、今回の判決を私なりに整理させていただきますと、基本的に新しい考え方を導入したということはないということになります。基本的には Campbell 事件判決、プリティ・ウーマン事件判決で出てきた“transformative”が非常に重要であるという考え方を踏襲しているということは言えるということになります。ですから、先例を覆すとか、覆さないまでも先例をある程度横に置いておいて新しい考え方を導入したというレベルのものではないと思われます。

（スライド 30、31）ただ、従来の判決に比べて特徴的なところは幾つかあります。これは前回もお話ししましたように、従来の裁判例というのは基本的には第 1 要素、第 2 要素、第 3 要素、第 4 要素と、フェアユースの第 1 要素「利用の目的」、第 2 要素「利用される著作物の性質」、第 3 要素「利用する部分の量と実質性」、第 4 要素「利用が著作権者の市場に与える影響」というのを順番に見ていくわけですね。

従来のほとんどの裁判例は 1、2、3、4 という順番で見てきたわけなのですけれども、今回はそうでなくて、先ほどもお見せしたように、第 2 要素、著作物の性質から検討を始めているというのは、非常に珍しいケースだと思います。そこに一つ特徴があるということです。この特徴をどう評価するかというのは、いろいろ議論のあるところだろうと思います。

一つは、プログラム著作物に特有だという考え方もできると思います。プログラムの著作物についての過去の判決、例えばリバースエンジニアリングに関する控訴裁レベルの判決にも第 2 要素から検討しているものがありますので、そういう点では、プログラムの著作物という、今となっ

てはそれほど違和感がないのかもしれないのですけれども、著作権の世界にとってはかなり違和感のある、若干異質なものを取り込んでしまったというところがありますので、プログラム著作物の性質というのが最初に検討されるということはあり得る。逆に言えば、プログラム著作物ゆえだという考え方はあるというふうに思います。

ただ、一つ言えるのは、従来1、2、3、4と、特にプリティ・ウーマン事件が1、2、3、4と検討して、しかも第1要素のところでtransformativeいうことによって変容力があると言ってしまうと、第2要素も、第3要素も、第4要素もそれに影響されて、結果的にフェアユース有利と働きやすい判断をしてきたという傾向がありました。そこのところは、順番が入れ替わって自由に検討できるというような感じを最高裁が示したと。少なくとも1、2、3、4という順番には最高裁はこだわっていないということが示されたということになると、これによって4要素の相対化というのが進む可能性もありまして、そういう点では若干時間を置いてみてみると、いろいろな影響があるかもしれないと思っています。

（スライド32、33）それから、第2要素のところですけれども、先ほども申し上げたように、従来、第2要素というのは非常に軽く扱われてきました。というのは、ほとんどのものが創造的、クリエイティブなエンターテイメント系の著作物が問題になることが多かったので、第2要素を重視してフェアユースに不利というのを強く出してしまうと、基本的にフェアユースは成立しないということになってしまいますので、ここのところは従来余り重視されなかったのですが、今回ここが非常に重要視されたということは、プログラムの場合はフェアユースに有利に振りましたけれども、非常にクリエイティブなエンターテイメント性の高いようなものになると、今後逆に、フェアユースに不利というふうに働きやすいようになる可能性もあると思います。多分、後半で石新委員からお話があると思いますが、絵画とか、現代芸術とか、パロディといったものに関して第2要素を重視するということになると、そういう価値のある芸術作品を簡単に利用させて良いのかというような議論をしやすくなる可能性があるのかなというふうに思っております。

第1要素のところですけれども、変容力があるというのが最重要であるということは今回の判決でも何も変わらなかったのですけれども、ただ、変容力があると言われたケースがパロディの事件だったわけですが、実はこれは目的と性格が違うということだけではなくて、プラス、表現自体が変わっているわけです。表現に手が加わっているという事案だったわけです。

過去に検索エンジンの事件というのが控訴裁レベルでありましたが、これは目的に変更があると。すなわち、画像のようなエンターテイメント系のものをインターネット上の情報の道しるべにするという点で目的の変化があると。しかし表現自体は変わっていない。画像は画像でそれがサムネイルになっただけということで、表現自体は変わっていないというのが、これは控訴裁レベルまでしか行かなかったですけれども、今回最高裁が初めて目的や性格の変更というプラスはあるのだけれども、表現自体は基本的に手が加えられずそのまま利用している。それであってもフェアユースになると正面から言っているという点では、一つ、変容力という言葉の意味を探究していく上でも重要かと思います。

“transformative”は、日本語では「変形的」などと紹介されることもありますけれども、見た目が変わるということはtransformativeであることにとって必須ではないというのは、アメリカの学説でも紹介されていて、地裁、高裁レベルでそういう判決も多数あったのですけれども、最高裁が示したのは飽くまでパロディの事件で、目的が変わっているプラス表現自体にも手が加えられていた、という事件だったので、本当にそこまで言って良いのか、見た目が変わらなくて良いのか、といったことは疑問として最後に残っていたのですけれども、そこは最高裁もこだわらないということがはっきりしたと言えるのだろうなと思います。

（スライド34、35）第3要素のところですけれども、これは従来どおりですが、量と質とを合わせて判断していくというのが今回確認されたということになります。そうすると大きいのは、リバースエンジニアリングや検索エンジン、機械学習など、全部複製しないと目的を達成しないというものについてフェアユースを認めやすくなるという影響はあるのだろうと思います。

それから、第4要素のところは、先ほども申し上げましたけれども、下級審は、変容力があれば市場代替性は低く、市場への害も少ないということを言ってきたのですが、今回の判決自体は、そこについて直接には言っていないのですね。言わなかったのはなぜかというと、どちらかというと、陪審員がある程度、影響がなかったということを既に事実認定として出していますので、それを尊重したということがあるので、こういう推定のようなことを言わなくても済んだという

ところはあるかと思います。
ただ、先ほど申し上げたように、大きな 2 つのポイントというのは、著作権法が保護すべき利益に影響があったのかどうかということをかなり問題にするということで、Oracle が言っている利益というのは、著作物そのものの利用から来る利益ではなくて、API に馴染（なじ）んだプログラマーがたくさんいて、それによって Java コミュニティという大きなマーケットがあるという、そういう意味の利益ではないのかと。で、それは著作権が別段直接保護するものではないというようなことが前提にあるのと、それからもう一つは先ほど申し上げたように、潜在的な市場というのを理論的な市場と解すべきではないということも言っている。
これは余計なことですが、日本の制限規定では最近、「権利者の利益を不当に害する場合には権利制限は適用されない。」というのがあるのですけれども、これの解釈にもかなり重要な示唆があって、そもそも、ありとあらゆる利益を考えて、そこに影響があったら駄目だという話ではないと。飽くまでも不当な場合だけというふうに、利益への害というのも、規定の趣旨、フェアユースの趣旨、権利制限規定の趣旨に照らして判断していくべきという点は、通底するのかなと思います。

（c）〔参考〕「巡回区判例による遅延現象」について

（スライド 37）最後ですけれども、先週お話ししたように、今のところ、下級審のほとんどの裁判例で、Google v. Oracle が引用されていないという問題はあります。そういう点では現時点における影響力は小さいと言って良いかと思います。
ただ、これについて一つだけ申し上げておくと、これは私が勝手に名前を付けているのですけれども、「巡回区判例による遅延現象」というのが実は見られます。最高裁判決が出たからといって、特に下級審、地裁の判決がころっとひっくり返るかというと必ずしもそうはいかないのですね。というのは、最高裁判例にはもちろん従わなくてはいけないのですけれども、各地裁というのは巡回控訴裁の判例を前提に判決をくみ上げていくわけですね。
そうすると、最高裁が直接自分が参照すべき巡回控訴裁の判決をひっくり返してくれたのであればそれはもう無視して良いということであっさり無視するのですけれども、巡回控訴裁が出している判決と最高裁が出した判決との関係がはっきりしないと、やはりどうしても、最高裁の判決からするとさほど重要でないのではないかとはたから見ていると思うような巡回控訴裁の判決も、地裁は結構引用するのですね。その結果、どうしても最高裁の判決の影響というのが巡回控訴裁レベルに上がって、巡回控訴裁自身が、最高裁の判決が出たので自分が今までやってきたことは軌道修正しますということが出ないと、なかなか軌道修正がされず、時間差が生じます。最終的には変わるのですが、時間差が数年かかっていくのですね。
例えば、従来で言えば、プリティ・ウーマン事件が出たときも、全体がプリティ・ウーマン事件の transformative という流れになっていくのに何年かかかるのですよ。その前はやはりずっとソニー事件の考え方というのが非常に支配的で、それが入れ替わっていくまでにかなり時間がかかるのですね。
したがって、そういうことがありますので、Google v. Oracle も今の時点では影響力は少ないのですけれども、このままずっと続くかどうかはもう少し見ないと分からない。あと数年かかってじわじわと切り替わっていくということもあるかと思われます。
そうは言っても先ほど申し上げたように、基本的な線はプリティ・ウーマン事件、Campbell 事件と変わりませんので、それほど大きな変更はないかなと思います。ただ、先ほど申し上げた幾つかの点で、新しい基軸を少し出しているところ、注目すべき点がありますが、そういう影響が数年かかってじわじわと、特に控訴裁レベルに影響し始めたときに、地裁の判決も雰囲気が変わっていくということはあるだろうと思っております。

b　AWF v. Goldsmith の概要（石新委員）（第 4 回 資料 2「AWF v. Goldsmith の概要」）

【山神】　奥邨委員、ありがとうございました。奥邨委員には、特に Google v. Oracle のフェアユースの判決としての特別性みたいなところについていろいろお話していただいたと思いますけれども、引き続き石新委員に資料 2 に基づいて御報告をいただきたいと思います。

(a) AWF v. Goldsmith の概要

【石新】　はい。では、始めさせていただきます。

奥邨委員の御報告に続きまして、The Andy Warhol Foundation とゴールドスミスという方の裁判について、概要を簡単に御紹介させていただきたいと思います。Google 判決がこの事件に大きな影響を与えておりまして、現在、アメリカの最高裁に係属中の事件となっています。それゆえに、奥邨委員の御報告に続けて、その後ということで、少し参考情報として御紹介させていただきたいと思います。

（スライド 2）ゴールドスミスさんは写真家の方でして、御覧いただいて左側なのですけれども、ヴァニティ・フェア誌にこの写真の利用許諾をしまして、ヴァニティ・フェア誌に 1984 年、プリンスの肖像写真が掲載されました。アンディ・ウォーホールという芸術家は、それに基づいてプリンスの肖像を描き、それが雑誌に掲載されました。それが今御覧いただいている右側の雑誌ということになります。

（スライド 3）ウォーホールはライセンスを受けて描いたわけですけれども、その肖像画以外にも同じ写真を使って似たような肖像画をゴールドスミスさんの知らないところで描いておりました。ウォーホールが死んだ後、ウォーホール財団が、ウォーホールが描いていた肖像画を、「プリンスシリーズ」として、ライセンスを行っておりました。

（スライド 4）ゴールドスミスは、2016 年にプリンスが亡くなった後になって初めて、自分の写真を利用されているプリンスシリーズというものの存在を知って、これは自分の写真の著作権を侵害するものであると警告を発しました。そのために、ウォーホール財団が自分から著作権侵害不存在確認の訴えを 2017 年 4 月に起こしたというのが事件の発端となります。判決に引用されているのですが、こういった形の肖像画をアンディ・ウォーホールが描いて、それをライセンスし続けてきたということです。

(b) 事案の概要

（スライド 5）裁判の概要なのですけれども、2019 年 7 月にニューヨーク州南部地区連邦地方裁判所は、ウォーホール作品が、プリンスを生きた人間として捉えている写真以上に、スターとしてのアイコンとしての表現になっていて、ゴールドスミスの写真の創作的部分はほとんど捨象されているということから、プリンスシリーズの制作について、写真のフェアユースであると判断いたしました。これに対してゴールドスミスが控訴したということになります。

(c) 第 2 巡回区控訴審判決

（スライド 6）2021 年の 3 月 26 日に第 2 巡回控訴審判決が出まして、結論を逆転させます。写真の利用をフェアユースとした原審を破棄、第 1 要素については両者が広い意味で視覚著作物として創作されたというだけでなく、同一人物の肖像でも全く同一であると。それから、全て二次的著作物とは言わないけれども、一部は写真を使って高コントラストでスクリーン印刷された異なる方式で描いていると。そこで、原作品を変容的に利用したのではなくて二次的著作物に該当すると判断しまして、第 1 要素についてはフェアユースに不利なものであると判断しました。

（スライド 7）この点、第 2 巡回区ゆえにという論点があるわけですけれども、同じ第 2 巡回区において、アプロプリエーションアートに写真を利用したケースが 2013 年に出ております。これもいろいろ議論があるところなのですけれども、Cariou v. Prince 判決というものがあるのですが、この事件においては、写真家である Cariou の原作品とは全く異なる美術上の目的を示唆する方法で芸術家のリチャード・プリンスが表現を付加しているということで、本件のアンディ・ウォーホールの利用とは違う判断をしたとしても、自分の先例に反するものではないというふうに、あえて Cariou 判決との違いについて言及しております。

（スライド 7）Cariou v. Prince 判決は、左側の写真が元々の写真家の写真で、右がリチャード・プリンスの利用形態というケースだったのですけれども、全部で 30 点ほどあって、25 点についてはフェアユースと認めて、5 点についてはフェアユースではないとしました。そこの線引き自体も議論があるところなのですけれども、そういった形でこのようなアプロプリエーションアートについて、第 2 巡回区がフェアユースを認めていたところに、この事件が違う判断をしたということも、注目されている理由の一つということになります。

（スライド 8）第 2 要素については、先ほど奥邨委員のお話にも出ていましたけれども、Google 判決とは違って、こうした一般的な芸術的な事件においては重要な役割を果たすことは少ないと。本件は創作性があり未公表であるということから、この写真の利用について第 2 要素についてはフェアユースに不利に働くとしています。

第 3 要素については、第 2 巡回区は、プリンスシリーズは本件写真の創作的な部分をそぐのではなくて、むしろ部分的に増幅させているというふうに判断して、これについてもフェアユースについて不利であると判断しました。

（スライド 9）第 4 要素ですけれども、ミュージシャンの肖像写真を典型的な二次利用のためにライセンスする市場というのは、実際にヴァニティ・フェア誌がライセンスを受けたように、既に存在していると。ゆえに、ウォーホルの利用というのはそういった市場を侵害するものだとしています。このようなウォーホルの無許諾利用を認めれば、ミュージシャンや有名人の肖像写真の典型的な利用、二次的利用という市場を破壊してしまって、著作権者にインセンティブを与えようとする著作権法に反する結果を招くということで、これについてもフェアユースに不利であるとしました。

このように、どの要素を捉えても、アンディ・ウォーホール財団に不利だということで、フェアユースを否定いたしました。

これが Google 判決とどういう関係があるのだという話です。

(d) 判決後の経緯

（スライド 10）時系列を御覧いただきたいのですけれども、2021 年の 3 月 26 日に第 2 巡回区控訴審判決が出ました。そのわずか 10 日後ぐらいでしょうか、2021 年 4 月 5 日に Google v. Oracle の最高裁判決が出たということで、アンディ・ウォーホール財団は、同判決を受けて、同じ月の 4 月 23 日に、Google 判決に基づけば第 2 巡回区控訴審裁判所の判断は間違っている、判例違反だということで、再審理の申立てを行いました。

こういう手続があるのは日本と違っていて面白いなと思ったのですが、それについて、同じ裁判体である第 2 巡回区控訴審は、2021 年 8 月 24 日に判決を修正いたしました。判決を修正はしたのですけれども、アンディ・ウォーホール財団が Google 判決に基づいて主張した内容については受け入れず、判決は維持ということになっております。

その後、アンディ・ウォーホール財団が同年 9 月に控訴審に対して全員再審理(en banc)の申立てたところ 9 月 10 日に却下されたので、同財団は同年 12 月 9 日に上告受理申立てをし、そうしたところ 2022 年 3 月 28 日に上告受理され、同年 10 月 12 日に口頭弁論が開かれたという時系列になっております。

なので、恐らくは 2023 年の上半期のどこかで、10 月 12 日に行われた口頭弁論の結果、フェアユースについての成否についての判断が下されると思われるという意味で、今日御紹介させていただいた次第です。

(e) Google v. Oracle

（スライド 11）アンディ・ウォーホール財団が Google 判決に基づいてどのような主張したか、どのような点を捉えて自分たちに有利なものとして再審理の申立てをしたかという点です。

Google 判決に限らず、アメリカの裁判例は一般的にそうですけれども、一般的な規範を広く展開した上で、具体的な事件の解決をしているというところで、本件の Google 判決においても、フェアユースの解釈について、かなり概説的なといいますか、一般論を展開している部分があるのですけれども、その中に自分たちが Campbell 判決の理解をどのようにしているかについて述べている部分がありまして、ここに幾つか摘示したのですけれども、この部分をかなり重視して上告受理申立てをしている。当事者の書面を見てもこの部分をかなり強調しています。

先ほど奥邨委員からお話があったとおり、実際にフェアユースの判例を見返したときに、transformative と言いながらそのまま利用してフェアユースと判断されたのは、Google v. Oracle が最初ということになりますので、その点を捉えて、新しい意味やメッセージを付け加えるということさえ実現できれば、形はそのままであっても transformative なのだというところについてかなり強調しておりまして、特に最高裁判例がこの中段のところで具体的に言っていると思うのですけれども、コンシューマリズムについてコメントするために広告のロゴをそのままコピーする

ということであってもフェアユースになるということについて、具体例として全くそのままコピーしても transformative というような例を挙げていると思うのですが、アンディ・ウォーホールの利用は正にこういったものに当たるということで、アンディ・ウォーホール財団側は、Google 判決のフェアユース判例に違反すると主張しているということになります。

（f） 再審理

（スライド 12）再審理についてですけれども、2021 年 8 月に判決が出て、その 6 ページのフットノートにその経緯について記載されています。そのまま引用させていただいていますけれども、なぜ自分たちがそのような再審理を認めたかということについて第 2 巡回区は述べておりまして、Google 判決の重要性ということで、注意深くその意見について検討して判断を下すためにこの petition を grant したというふうに述べております。

（g） Google 判決の影響について

（スライド 13）その上で、では第 2 巡回区控訴審裁判所は Google 判決というものについて、どういうふうに読んだかということなのですけれども、結局、アンディ・ウォーホール財団の第 2 巡回区控訴裁判所が 3 月に下した判決は、4 月 5 日の Google 判決に抵触するという主張は認めず、むしろその判断は Google 判決に沿うとして、排斥しております。第 2 巡回区控訴審裁判所に言わせると、Google 判決は、フェアユース判断というものが極めて事例に即したケーススペシフィックな判決になるということを明確にしていると。ゆえに、コンピューターという非常に特殊な事例についてあのような判断をしたのだと。コンピュータプログラムの文脈でそのような判断をしたからといって、今回のような芸術的な作品についての著作権の問題にそのまま当てはまるものではないと。著作権の保護というのは、著作物が実用的なもの、機能的なものではなくて、芸術的なものである方が強いという一般論もあり、そういった著作物の違い、事案の違いということを考えると、Google 判決はむしろ第 2 巡回区が 3 月に下した判決を支持するものであるというふうに述べております。

英語の部分は第 2 巡回区が自分たちの判断に有利な部分として最高裁判決から引用した部分です。機能的な著作物であるコンピュータプログラムを扱う場合に、伝統的な著作権の概念を適用するのは難しいと、著作物の性質の違いというものを明確に意識して判断しているという意味で、Google 判決と第 2 巡回区の 3 月の判決は矛盾しないというふうに結論付けたということであります。

（h） Kienitz v. Sconnie Nation LLC, 766 F.3d 756 (7th Cir. 2014)

（スライド 14）最後は、第 2 巡回区の判決とは全く関係ないのですけれども、最高裁判決のフェアユースの判断がどうなるかということとの関係で、フェアユースの判断の仕方を統一する意味合いがかなり強く出てくるのではないかというふうに言われているようでありまして、その一例として、この Kienitz v. Sconnie Nation LLC, 766 F.3d 756 (7th Cir. 2014)という事件があります。同じような事例なのですけれども、左側の写真の著作権を侵害するものだとして、T シャツなどに利用された右側の利用形態がフェアユースかどうかが争われたケースです。

地裁では、当事者は、transformative であるか否かについて Campbell 判決に基づいて散々議論をしているところ、第 7 巡回区は、transformative という議論自体を遮断してしまって、それは著作権法 107 条の条文上の根拠を欠くものだと言って、一番重要なのはやはり第 4 要素で、結局、市場を奪うものか、市場代替性を有するものか、それともそうではなくて支えるものにすぎないのかといった、replacement になってしまうかどうかということがフェアユースの成否の決定的な根拠、基準になるべきであって、transformative が第 2 要素以下第 4 要素まで全てを置き換えてしまう、第 1 要素の transformativeness が一番重要だというような解釈は誤っているということで、transformative の概念を使うことに懐疑的な巡回区裁判所ということになります。

2013 年に第 2 巡回区で transformative だという Cariou 判決が出て、翌年の 2014 年の第 7 巡回区では transformative について懐疑的な判決が出て、それゆえに第 7 巡回区と第 2 巡回区とで判断が分かれている、circuit split があるということで、上告受理申立てがこの点でもされているのですけれども、このケースでは取られませんでした。

それから 9 年後の 2023 年になって、いよいよ最高裁がコンピュータプログラムではない文脈

において Campbell 判決で自分たちが言った transformative というものをどう当てはめるのか、どう利用するのかということについて、その判断の行方によっては circuit split になっている状態が統一されていくという、先ほど奥邨委員から下級審に浸透するまでに時間がかかるというお話がありましたけれども、いずれにせよ一定の方向性が示されるのではないかと注目されている状況であります。

この資料に「UCLA Netanel 教授の PPT より」と書いたのですけれども、UCLA の Neil Netanel 教授と昨年（2022 年）の 4 月にお会いすることがあって、そのときにいろいろな情報を頂いたうちの一部から御紹介したのですけれども、この事件のことをちょっとお聞きしたところ、Netanel 教授は、transformative が広がっていくというか、弾力的に動いていることについては肯定的で、不当に transformative という概念が広がり続けているというコロンビア・ロースクールのジェーン.C.ギンズバーグ教授に比べると対照的な考え方をされているのではありますが、Netanel 教授に「最高裁が取り上げるからいいですね。」というふうにお話したところ、かなり不安があるようで、著作権のミニマリストといいますか、余り大きく広く著作権を考えない判事だった、Google 判決を書いたブライヤーさんが、去年（2022 年）の 4 月に　Netanel 教授とお会いした時点で既に退任することが決まっていたと思いますので、彼が辞めた後の著作権判例の行方がちょっと不透明であるということと、構成メンバーを見ても、この事件でアンディ・ウォーホールが敗訴するというだけではなくて、transformative という概念が不当に狭められるというか、非常にたがをはめられてしまうのではないかということについて、むしろ懸念を持っているというようなことを当時話されておりました。

いずれにせよ、そういうことで、あちらでは結構注目されている判決ということになります。

（イ） 討議

【山神】　石新委員、どうもありがとうございました。

最後の部分をやや聞き逃してしまったのですが、transformative なものを狭めてしまうというのは、どの判決のことを指して発言されたのでしょうか。最後の部分なのですけれども。

【石新】　アンディ・ウォーホールのような現代芸術を利用するという形態において transformative は認められるべきだというふうに Netanel 教授は考えているのですけれども、それが明確に否定される、例えば、先ほどの 2014 年のケースでもそうですけれども、上告不受理という形で終わっているのであればまだ控訴審のレベルでいろいろあり得たところが、最高裁に取り上げられてしまったことによって、「こういった利用はライセンスを得ないとマーケットを侵害するものだとして著作権侵害になる。」ということが明確に述べられてしまった場合に、上告不受理であればまだまだ広がっていく余地があった現代芸術における無許諾利用といったものが完全に否定されてマイナスな効果を生むことになるのではないか、本来的には判例法においてもう少し広めに広がっていくような性格のものだと位置付けている学者やコメンテーターの方々が多い中で、例えば、フェアユースという判断をするのであれば、恐らくは、「95 年、96 年の段階で transformative と言ったのは、自分たちにとってはこういう意味だ。」ということを一般論として展開するのではないか、良い悪いは別論として、明確にこの範囲でしか駄目なのです、というかなり限定的なことが最高裁によって言われてしまうことにより、Netanel 教授からするとネガティブなインパクトが生じるのではないか。

そのようなことを心配されていた、という趣旨です。

【山神】　分かりました。要するにアンディ・ウォーホールの事件は今最高裁で係属中だけれども、そこで良からぬ結論が出るのではないかという懸念ということですよね。

【石新】　そういうことですね。教授個人の感覚ですけれども、そういう趣旨です。

【山神】　大変よく分かります。Kienitz v. Sconnie Nation LLC の事件は、上告不受理で終わったのですよね。

【石新】　そうです。circuit split だと言って上告受理申立てをしたようなのですが、2015 年に上告不受理となっています。

【山神】　分かりました。ありがとうございました。

もう 1 点だけすみません。Kienitz v. Sconnie Nation LLC の事件は、確か lazy appropriator というか、（スライド 14）左側が多分写真家の人が撮った写真で、それを加工すると。これは本当に先ほどの transformative なという話で、フェアユースに有利に働きそうだけれども、この判決自身

は割と何も考えずに流用してしまう人に対してものすごく厳しいことを言っていたかのように思うのですが、その辺り、何か議論などはあるのでしょうか。

【石新】　すみません、該当部分について現時点では確認できません。

【山神】　私の手元でも資料が出てこないのですが、私の記憶では、確か、コピペまではいかないのですけれども、使うことについてかなり否定的なことを言ったような気がするのですけれども。

フェアユースというのは、要するに lazy appropriator を保護するものではないという感じのことを・・・。だから、そういうふうに言ってしまって良いのか、あるいはそもそもこういう利用であればフェアユースなのだというふうに思ってしまう利用者からすると、そういうことを要求されても、わざわざ使っても良いはずのものを、使うために何かいろいろ手を加えて無駄なことをするのが必要なのかどうかというのが、昔、気になったところなのですけれども。すみません、クリアになりました。ありがとうございます。

大谷委員、どうぞ発言をお願いいたします。

【大谷】　本当に基本的なところを教えていただければと思うのですけれども、第 1 要素については、Google 判決は再実装というところを捉えて、transformative だという判断をしているので、第 2 要素から話を進めていたとしても第 1 要素を抜きに語ったものではないということなのですが、第 2 要素から説き起こしたということは、著作物の性質によって transformative の形態にも様々な振れ幅があり、プログラムのような場合には、再実装というのは多分アート系のものには余り考えにくいというか、もちろんプリティ・ウーマン事件のようなケースでは認めるのだと思いますし、著作物の性質と性質に応じた transformative、変形力の幅というのは、もちろん裁判の流れというか、時代の流れによって変わってきた面もあると思うのですけれども、今は米国の巡回区裁判所でおおむねどういうふうに理解されているのか解説いただけると全体像が分かりやすくなるかと思います。奥邨委員でも石新委員でも教えていただけると有り難いです。初歩的な質問で恐縮です。

【山神】　いかがでしょうか。先ほどから石新委員に大分お答えいただいているので、もし奥邨委員、いきなりの御指名で恐縮ですけれども、何かコメントがありましたらお願いいたします。

【奥邨】　著作物の性質によって transformative の幅が変わってくるかということですね。そこまで明確にそうした議論があるかというと余り記憶がないのですけれども、ただ、今回の判決を見ていても、それが直接影響するというものではないと思います。著作物の性質よりは、どういう目的で使うか、使い方の問題ということなので、それによって transformative の幅が変わってくるということではないと思います。

今回のものは、プログラムの中でも特に著作権の保護の核心から遠いのだということを言っていましたけれども、機能と結びついているというのは、全てのプログラムに共通した性質なので、そこで議論が終わってしまっては第 1 要素の検討にはならないのだと。今回の「使い方」というのはどういうものだというのをより詳しく見ていきましょうと。そこから先を具体的に見ないといけないと言っていますので。

したがって、プログラムだとか何だとか、著作物の特徴で議論が終わるのではなくて、具体的に何のためにどう使ったのかというところに踏み込んで見ていかないといけないと言っていますので、そういう点では、私自身は、この判決を読んだときの感想としても、そのことによって transformative 性が影響を受けるというふうには受け取らなかったかなと思っております。

【山神】　奥邨委員、ありがとうございます。今のお答えで、単なるプログラムというふうにざっくり切るのではなくて、対象著作物の性質といいますか、そこのところをもっと丁寧に読み込んで、今委員がおっしゃったようなところを考えて、そこから第 1 要素の方の話へ行くというのはあり得るのかなという気がいたしましたけれど、その辺りはいかがでしょうか。

今回ですと、利用されている API の更に宣言コードなど著作物と推定される情報の性質といったことをどの辺りに埋め込むのが良いのか、何かお考えはございますか。

【奥邨】　私自身がこの判決を読んで感じたのは、裁判所は第 2 要素が第 1 要素に影響を与えるということは考えていなくて、第 2 要素が、第 3 要素、第 4 要素の方に、特に第 3 ですけれども、影響を与えるという感じだったような気がします。

従来は、第 1 要素を最初に検討したら、それが第 2 要素、第 3 要素、第 4 要素の検討に全部影響を与えてくるということだったのですけれども、今回は第 2 が出てくることによって、第 2 が第 1 以外のものに影響を与える。特に第 3 要素に影響を与えると。それで、第 1 要素は第 3 要

素、第 4 要素の判断に影響を与えるということで、二重に他の要素に影響を与えているという感じに読みました。

第 1 と第 2 の間で影響し合うという感じでは余りなかったというふうに思いますし、従来も第 2 が第 1 に影響を与えることも「子が親に」みたいな力関係になっておりますので、そういう感じもなく、それをひっくり返すようなことを最高裁が言ったような感じはなかったと、私は理解をしております。

【山神】　ありがとうございました。そうしましたら、宮下委員、発言をお願いいたします。

【宮下】　はい、本日は非常に興味深いお話をどうもありがとうございました。

お話を伺っていて思ったのが、今回第 2 要素から入ったというのは、やはり著作物性が認められるための創作性が非常に弱いというか、創作的な特徴がさほどないものが利用されたという点において、やはり第 2 要素から考えないといけないと。創作性のレベルが低いようなものに関しては、商業的であったとしても、フェアユースが認められる可能性が高いのだ、みたいな考え方に根差すものなのかなという気がしたのですね。

アンディ・ウォーホールの事件や、もう一つ石新委員に御紹介いただいた事件を見ると、日本の著作権の判断ですと、創作的ではない部分、人の顔の特徴とか、創作者の個性が反映しているというよりは事実自体が共通しているというようなことで、本質的な特徴を直接感得できるようなものではないというような見方をすることもあるのかなと。

日本では創作性のレベルがアメリカの「オリジナリティ」より若干高いのかなという理解を持っているのですけれども、日本であれば創作性の判断で結論が出されるようなものを、アメリカの著作権紛争ではフェアユースのところで創作性のレベルも考慮した形で判断されるというような特徴があるのかなとも思ったりしたのですけれども、そういう見方についてどういうふうにお考えになるのか、お話を伺えれば幸いです。

【奥邨】　奥邨です。今回のケースは委員がおっしゃったような形で触れたという見方もできると思います。

ただ、一方、従来のアメリカは、日本で言うと「それは著作物ではないでしょう。」というような部分についてまで著作物性を認めるものもあります。例えばキャラクターですね。日本だとキャラクターの絵柄ではなくて性格付けとか場面設定というのはかなり狭く、「それはアイデアの世界ですよね。」と言ってしまう部分が多いと思うのですけれども、アメリカの場合は、表現になっていなくても、ある程度特定されているものについては、キャラクターも著作物として保護されるという考え方があるわけなのですね。そのようなことで一旦著作物性を広く認めて、しかし「それは困るよね。」というものについて、フェアユースと言ったケースが幾つかあるのですね。例えばコメディなどで似ているといって問題になったときに、日本であれば多分著作物性はないというところで切っていたところがありますので、したがって、「広い、狭い」のところで、多分、日本とアメリカで、著作物性が違うことと権利制限規定が違うことで、結果として同じようになるようにバランスを取っていると、好意的に見るとそういうことなのかなと思っております。

【宮下】　ありがとうございます。

石新委員はどのようにお考えでしょうか。

【石新】　私も、宮下委員がおっしゃったように、このケースを見たときに、日本であればやはり本質的特徴を直接感得できない、と。間接的にちょっと似ているなという感じるぐらいで、この人が本人だなという肖像的な意味合いが分かっても、写真の持っている創作性の根拠となるような特徴部分が感得できないという形で切れるかなと。特に Kienitz v. Sconnie Nation LLC という最後に御紹介した第 7 巡回区の事件などは、日本であれば別にフェアユースや権利制限といったところを議論せずに、写真の本質的特徴が感得できないと言って写真家が負けてしまうというか、利用を認めることも可能なのかなというふうに考えていましたので、その点、宮下委員と同じような感想を持っておりました。

【宮下】　ありがとうございます。

【山神】　今村委員、御発言をお願いいたします。

【今村】　すみません、手は挙げましたが、今ちょうど宮下委員から質問があった点と同じような点をお伺いしようと思っていました。多分そうなのだろうなということで理解ができました。しかし、せっかくなので一応お伺いします。

石新委員の方で御紹介いただいた写真の事例、石新委員の資料 6 ページのところで、「プリンスシリーズが全て 2 次的著作物に該当するものではないが、（中略）原作品を変容的に利用したものでなく 2 次的著作物に該当する」とあるのですが、ちょっとよく分かりませんでした。アメリカ法の「変容的」ということをよく理解できてない部分があるのですが、日本法の頭で考えると、変容的に利用すると、単なる複製ではなくて二次的著作物とか翻案物になるのかなというイメージがあるのですが、変容的に利用したものでないと二次的著作物に該当するということは、二次的著作物の捉え方が日本法とは違うということなのでしょうか。アメリカにおける二次的著作物の作成というのがどういうものを指すのかということを確認させていただければと思います。

【石新】　分かりにくい表現ですみません。間違っていたら奥邨委員に直していただければと思います。

私が理解している限りですと、特に transformative の概念を理解するときに、Jane Ginsburg 教授のものを読むとすごくよく分かる、自分で分かったつもりになるのですが、要するに、transformative という著作権法の条文にない用語がパロディのケースで使われて、それがどんどんいろいろなことに使われ過ぎているという認識を示されています。そうすると結局、アメリカの著作権法上、原著作物に何らかの創作的なものを付け加えて何か作っても良いわけですが、作ったらそれは二次的著作物、derivative right が働いて、そこは日本と同じですが、原著作者の同意が必要なのではないかということで、結局、フェアユースの transformative と二次的著作物、derivative right の守備範囲はどこで線引きするのかいうところが、条文上にない概念だったので、Campbell 判決の後、「どこで線引きするのか。」と、下級審がずっと悪戦苦闘していた。transformative と言えば、そこは二次的著作物として承諾を得なければいけないものではなくて、権利制限として無許諾利用が可能であると。ただ、transformative と言えない場合は、そこは結局原著作物を利用して自分の創作物を作っているので、それは二次的著作物になるという意味で、違法と合法の線引きの結果だけを部分的に取り出しているので分からなくなってしまっているのですが、結局プリンスシリーズとして彼が肖像画を描いたものについて、全てが全て、元とした写真の二次的著作物、要するに著作権侵害になるようなものではないのだけれども、今回違法だと判断した、本件写真をベースにして描いた肖像画については、変容性が認められないから、それは二次的著作物であり、承諾なく二次的著作物を創った、ゆえに違法ということで判断した。分かりにくくて恐縮ですが、許諾なく利用することについては原則二次的著作物の侵害となる。

ただ、それが transformative だと言えるときにはフェアユースとなって、そもそも承諾の要る二次的著作物ではなく、変容物として許容されるということなので、「承諾なく作って良い二次的著作物」みたいな概念になると思いますが、その二種類をここで使い分けたいというか、言い分けたい。違法な場合と合法な場合、ということです。

なので、最高裁でも論点になっているのは derivative right の範囲と transformative の範囲の線引きを、どこでどういう基準でするのかと。何かを意味付けしたり、メッセージを付けたりしたというのは、小説を映画にする際にも新たなメッセージや意味が付加されたりする場合があるが、それを変容していると言ってフェアユースだという場合はない。では、どういう場合が transformative でどういう場合が derivative right の守備範囲なのかということが、ずっと論点になっています。

そういうこともあって、第 2 巡回区は、derivative right との守備範囲の違いを意識してこの言葉を使って判示している。その部分から一部を私が取り出しているので、こういう表現になっております。すみません、分かりにくくて。

【奥邨】　今のところ、1 点だけ追加で申し上げると、実は同じ第 2 巡回区が、今から 20 年ぐらい前、1998 年の判決から、正に今の点で微妙な言葉遣いをしていまして、transformative な作品とそうではない二次的著作物との区別、というような言い方をしているのですね。私たちの割り切り方からすると、まず二次的著作物になるかどうかという議論があって、その上で抗弁として transformative かどうかの議論が出てきそうに思うのですが、その辺りが余りきれいに分かれていない。

transformative な作品というのは、概念的に言うと、元のものを変更して作ったものなのだけれども、transformative なものと言っているのは権利が及ばないものだし、一方で二次的著作物、derivative と言っているのは権利が及ぶもの、というザクッとした議論をしていて、今私たちがし

ているように一つ一つ詰めて考えているという感じでもなさそうな気がしています。それを随分昔、20 年前の判決でも言っていたのですが、多分その辺が余り変わっておらず、今、石新委員がおっしゃったように、今回の判決でも第 2 巡回区ですから、自分の過去の判決を引きながら、こういう区別をして言っている、そこの区別が実は難しいのだと、そういう前提に立っているのだろうと思います。

【今村】　何となくわかりました。日本法とは随分考えのプロセスが違うようですし、フェアユースが非常に万能のようなものなので、それがばっと先に出てくると、結局のところ二次的著作物であろうが何だろうが、余り関係なくなってしまうというのも背景にあるのかなとか、いろいろ考えました。ありがとうございました。

【山神】　ありがとうございました。岩原委員が手を挙げられましたので、どうぞ発言をお願いいたします。

【岩原】　いろいろ教えていただきまして、ありがとうございます。

前回議論されたことと関係するかもしれませんし、初歩的な質問で恐縮なのですけれども、今日の第 2 要素と第 1 要素について、プログラムについての考え方を確認させていただきたいと思います。

奥邨委員の資料の 19 ページで、今回の判決ですと、機能的かそうでない部分かというのがある程度重要で、名前をどういうふうにラベル付けするかというところはどちらかというと創作性があって、プログラムの構造、今回の API であれば、オブジェクト指向であれば、その形についてどういうふうに用意するかとか、もう少し簡単に言えば、例えば「pi」という名前なのか「enshuritsu」とローマ字で書くのかというような名前付けは別として円周率を用意しておくとか、ネイピア指数を用意しておくとかというような機能的な部分と分けたときに、機能的な部分をまねていたとしても、どちらかというと変容力はなしの方向になるけれども、「enshuritsu」とか「pi」というような名前付け、ラベル名を付けるところは変容力ありというか、第 2 要素で言えば重要部分になるといった理解でよろしいでしょうか。

【山神】　今のところですが、前回議論をしているときに野山さんからコメントいただいたかと思いますが、関数名などいろいろな名前をユニークに付けるのは、本来自由に付けられるわけですが、エイヤと決めてしまわないといけないと。しかし、一旦決めてしまってそれを標準化している以上、それと互換のものを作るときには、もうおよそ選択の余地がなくなるという整理だったと思います。奥邨委員もそういうふうにおっしゃっていたかと思いますが、いかがでしょうか。

【奥邨】　私のレジュメのまとめ方も良くないと思うのですけれども、（スライド 19）まずここのところで見ますと、実装コードの場合は、効率とスピードを両立させてコンピュータを稼働させられるかなどの点に創造性を発揮したと言っている一方、宣言コードの場合は、名前を覚えやすくする点に創造性を発揮したのだけどれも、実はこの点は、今回の宣言コードの価値ではないのだと。発揮した宣言コードの創造性は価値になっていませんよ、というふうにつながっていくのですね。では宣言コードのどこに価値があるのかというと、それを皆が一生懸命覚えたところに価値があるのだと。創造性を発揮したかもしれないけれども、この創造性は著作権の世界で守るべき価値では全然ない、少なくとも今回の中ではそこに重きを置く必要はありません、という流れになっていくということなので、ここにある「創造性」は、「creativitty」を訳していまして、著作権法の「originarity」、創作性を意味することがではないから「創造性」と訳しているはずなのですけれども、著作権法上評価される「originarity」かどうかは別にして、一応工夫はしたと。工夫はしたけれども、それは今回の評価の中では取るに足らないものだとなっていますので、どちらかと言えばマイナスに評価されてしまっているとお取りいただいた方が良いのかなと思っています。

【岩原】　ありがとうございます。そうしますと、機能性が高まれば高まるほど、著作物的な価値が低くなるといいますか、高くは評価されないという理解は、よろしいでしょうか。

【奥邨】　一般論としてはそうなのだろうと思うのですけれども、今回そこは直接、中心には言っておらず、飽くまで宣言コードだけを議論しているということです。しかも宣言コードの部分が機能だということを前面に言っているというよりは、体系化とか、よりアイデアに近い部分という言い方をしていたのではないかと思います。機能というのはどちらかというと、動いているという意味において、効率とスピードを両立させてコンピュータを稼働する実装コードの方だったというふうに思います。

【山神】　よろしいでしょうか。
【岩原】　はい、結構です。

イ　日本法の下での権利制限規定の適用可能性

【山神】　今のお話のところはちょうど次の伊藤委員の御報告で、日本法はフェアユースがございませんので、ではどうしましょうかという御報告をいただけます。そこで多分つながってくると思いますので、伊藤委員、どうぞよろしくお願いいたします。

(ｱ)　報告「―API の法的保護―日本の著作権法のもとでの権利制限規定の適用可能性」（伊藤委員）（第 4 回　資料 3「―API の法的保護―日本の著作権法のもとでの権利制限規定の適用可能性」）

【伊藤】　私の方では、日本の著作権法の下で仮にこの Java API が著作物であると認められた場合に、権利制限規定を適用することができるかということについて発表させていただきます。

a　テーマ

（スライド 2）テーマはここに述べたとおりです（「仮に Java API に著作物性が認められた場合、フェア・ユース規定がない日本の著作権法のもとで、権利制限規定の適用によって Java API を適法に利用することはできるか。」）。

b　著作物性がある場合の権利制限①

御承知のとおり、日本にはフェアユース規定がないわけですが、幾つかの個別的な権利制限規定があります。

（スライド 3）それを全部というわけではないですが、思い当たるところとしては、最初に 30 条の 4、平成 30 年に導入された非享受利用のところが一つ考えられます。こちらの規定については「思想又は感情を自ら享受し又は他人に享受させることを目的としない場合には」利用することができるという規定があるのですが、第 3 号では「プログラムの著作物にあつては、当該著作物の電子計算機における実行を除く」などのちょっと気になる記述などもありますので、この「享受」について、非享受利用に当たるかというところを少し検討していきたいと思います。

（スライド 4）この「享受」という言葉については、文化庁の立法当時の解説からひも解いていきますと、やや抽象的な言い方になっていますが、「著作物等の視聴等を通じて、視聴者等の知的・精神的欲求を満たすという効用を得ることに向けられた行為であるか否かという観点から判断」するとされています。

（スライド 5）続いて、30 条の 4 におけるプログラムについての説明があります。いわゆるリバースエンジニアリングがこの非享受利用に当たるかどうかということが当時から議論されていたわけですけれども、その文脈において、プログラムにおける「享受」の意味が議論、解説されています。「「享受」に該当するか否かは、当該プログラムを実行等することを通じて、その機能に関する効用を得ることに向けられた行為であるかという観点から判断される」と。プログラムにおける享受というのは、機能に関する効用を得るかどうかということで判断されるべきものだと述べられています。

（スライド 6）文字ばかりなので、全部は読み上げませんが、結局プログラムについては、プログラムの機能に関する効用を得ることに向けられた利用行為であると考えられると。リバースエンジニアリングの場合はプログラムの調査解析を目的としているので、機能を享受することに向けられた利用行為ではないから、非享受利用に該当するものと考えられる、との説明がなされています。

（スライド 7）これは飽くまでリバースエンジニアリングについての検討したものですが、翻って、今回の API の宣言コードについては、結局何をするかというと、API の宣言コード部分を、第三者たる Google がコピーして、それがコードの一部を構成するということで、これの場合は結局プログラムの機能に関する効用を得ることになるのではないかと考えられますので、やはり非享受利用には当たらないのではないか、つまり 30 条の 4 によって権利制限に当たり、著作権侵害にならないということは言えないのではないかと考えられます。

c　著作物性がある場合の権利制限②

（スライド 8）もう 1 つ、権利制限規定として適用可能性があるかないかというところで、これはずっと以前からある「引用」ですね。条文自体は非常にシンプルで、「公正な慣行に合致するものであり、かつ、報道、批評、研究その他の引用の目的上正当な範囲内で行われるものでなければならない。」。

（スライド 9）この規定について、これまで、実務上は、モンタージュ事件の判例を引いて、いわゆる明瞭区別性と主従関係があること、というようなことが言われていました。これは実際には旧法下の事件であるとか、傍論部分であるというようなことはもちろん言われているわけなのですが、この 2 つの要素を使った幾つかの裁判例があることに即して考えますと、API の宣言コード部分を取り出して、そして実装コードを自分で書くことにした場合には、少なくとも、見る人が見れば、宣言コードと実装コードは明瞭に区別して認識することはできそうだ、と。

実際に、本件で言うところの Google に当たる第三者が実装コードを自分で書くと、この右側の例で言うと「{ if(x>y), return x else return y }」の部分で、この図では非常に少なく見えるのですが、実際のコードで言うとこちらの実装コードの方がボリュームとしては圧倒的に多いわけなので、実装コードを作成していれば、そちらが主であって、宣言コードは従であるということは一応言えるかもしれない。引用という言葉のイメージからすると余りしっくりこない部分もありますが、明瞭区別性と主従関係という以前から言われていた基準に照らすと、満たすような気もします。

（スライド 10）もう 1 つ検討したのは、その後、比較的新しい事例である絵画鑑定書事件で、明瞭区別性や主従関係という基準を用いずに判断したわけなのですが、引用としての利用に当たるか否かの判断においては、条文にある「著作物を利用する側の利用の目的のほか、その方法や態様、利用される著作物の種類や性質、当該著作物の著作権者に及ぼす影響の有無・程度などが総合考慮されなければならない。」ということで、既に言われていて本日も御説明をいただいたフェアユースの 4 要素にちょっと重なるところもあるということで、実際に API の宣言コードを利用した場合について、この絵画鑑定書事件で挙げられていた要素を踏まえて検討してみると、目的は、最高裁判決も言っていましたが、モバイル用の開発環境を整備するためですし、PC ソフトの開発者のスキルがそのままモバイル開発でも生かせるようにというような目的があり、方法や態様については、どう当てはめて良いかちょっと迷うところではありますが、実際には実装コード自体はコピー、利用していなくて、宣言コード部分のみであって、創作性が必ずしも高くない部分を使っているのみであるとか、プログラムの分かりやすくて機能性が求められる部分、こうでなくてはいけないという決まり事の部分を利用しているということ。そして、著作権者に及ぼす影響ということで、鑑定書の事件とは余りにも違っているので同じように考えるのは難しいかもしれませんが、先ほど御説明もあった第 4 要素にちょっと近いものなのかなと考えると、やはりこちらも、Oracle、Sun とは、モバイルと PC ということで市場は全く異なるということで、絵画鑑定書事件等で言われた基準に照らしても、引用が言えるかもしれないと。

判決が出た後の 2021 年の（SOFTIC の）セミナーで担当して発表したときには、権利制限規定の適用は難しいのではないかと申し上げてはいたのですが、改めて明瞭区別性、主従関係とか、あるいは絵画鑑定書事件を読んで当てはめてみると、意外に引用もいけてしまうかもしれないなと、今日のところは思っております。

（イ） 討議

【山神】　ありがとうございました。今の御報告、かなり刺激的だと思いますが、御意見、御質問ありましたらどうぞ。岩原委員、お願いいたします。

【岩原】　ありがとうございます。引用でいけるというのは今非常に驚いていまして、いろいろ教えていただきたいと思います。

主と従と量の関係でということかと思ったのですが、そうしますと、例えばいわゆるライブラリ的なもの、汎用的なモジュールがあった場合、それを丸ごとコピーして、そのモジュールを使った実装コード、更にそれを利用したルーチンを大量に記述することはよくあると思いますが、そうしますと、引用されるモジュール、ライブラリー的なものは、当然、明瞭に区別できて、量も

全然大きさが違うということになると、今の API の場合の御説明の引用の例と非常に似てくるのではないかと思います。この場合もやはり引用でいけるというお話になるでしょうか。それとも今の、いわゆる独立したモジュールをサブルーチン的に使う場合と API の利用の場合は何か違う点があるということなのでしょうか。

【伊藤】　今おっしゃった例は、Java で言うところの宣言コードだけをコピーして、実装コードを自分たちで書くケースではなく、通常のライブラリーの利用のケースを挙げられたと思いますが、同じかどうかと言われたところの事例がよく分からなかったのですが。

【岩原】　失礼いたしました。今回の Google 事件のようなものは、API の結合部分的な取決めの部分だけを宣言していると。それを取り込んで、その中身は別に書くというものですよね。そのときの宣言部分を取り込んで、中身、実装コードを実際に書く。

　そうすると、宣言部分は、人が作ったものを自分のプログラム上でどこか頭の方に書いておいてその中身を別に書くという点で、呼出しに近い形だとは思うのですけれども、それとは別に、資料 9 ページに出ている Method Call とあるところにあるモジュールが、そのまま書かれていたのを丸ごとコピーしてきて書く場合ですね。第三者が作ったモジュールそのものをコピーして、どちらからどちらかに呼ばれるという形は方向は多分逆になると思いますが、逆だとしても、使われ方や主従の関係、独立性というところはすごく似ているのではないか、というのが質問です。

【伊藤】　そうですね。ただ、モジュールの中身自体は、その場合でも他人が使ったものをそのまま使っていて、利用者はコールしているだけと、そういう意味ですよね。

【岩原】　そうですね。これを引用と位置付けるとすれば、第三者のモジュールそのものが引用対象になっていて、それをコールしている側、コピーした方が実装コードとしてこれから新しく作る部分がモジュール部分を呼んでいるという点では、ここで書かれている図だと、主従関係に似てくるのではないかというところです。

【伊藤】　そうですね。主従関係は似ているけれども、その場合は呼ばれる側のモジュールを著作権者に無断で使った場合に引用で逆に救われてしまうとすると、他人の作ったモジュールは使いたい放題になるのではないか、という問題意識でしょうか。

【岩原】　はい。この枠組みでいくと、その先にはモジュールなども同じような文脈で言われてしまうと、日本法で考えると、結局主従と分量、そして「不当に」という、今日の冒頭であったような要件さえクリアしてしまえば、いわゆるライブラリの中から部分的に引っ張ってきて入れてしまったとしても、何でもできてしまうということになりはしないかというのが最終的な問題意識になるのですけれども。

【伊藤】　呼ばれる側のモジュールそのものをローカルにコピーしてしまうと、分量的に多くなってしまいますし、それ自体、主従関係と言えるかというのは微妙な感じはしますが、単に宣言部分を添えてコールするだけであれば、やはり同じように著作権法的な問題はないのではないかと思いました。私が問題意識をうまく理解していないかもしれませんが。同じ状態になるということはよく分かります。

【岩原】　そうしますと、違いは、宣言コードの場合は一つ一つの塊が通常数行で終わることが多いところ、別のモジュールというか、ルーチン的なものだと、一つ一つがそこそこの長さになるという、そこの部分の違いでしょうか。結論として、多分、伊藤委員も違う結論になるというお話かなと今聞いていて思いましたが、API の場合と、ほかの、いわゆるルーチン的なものを持ってきてコピーする場合との違いを分ける分水嶺はどこなのかというところなのですけれども。

【伊藤】　実装コードを全部コピーしているかどうかの違いかなと思ったのですが。外のルーチンを呼ぶということは、呼ばれたときに実行できないといけないので、その実行部分も合わせてコピーしてしまっているとすると、それはこの API の事件とは違うわけですし、実装コード部分は相変わらずコピーはしないで単に呼び出しているだけであれば、確かに一緒かなと思います。

【岩原】　ありがとうございます。

【山神】　岩原委員が出された例というのは、恐らく、資料 9 ページですと、Sun の Java API があって、多分伊藤委員がお答えになったように、ライブラリの場合宣言コードだけではなく実装コードまで全部含めて使うことになるので、それは引用としてはかなり厳しいのかなと。

　逆に、今回の場合だと、問題となっているのは Java API の全体像の中の宣言コードの部分のみです。この 9 ページの図の右端にはありませんが、実際には Dalvik という Android の仮想マシンがあって、そこが持っている API というものがあって、そこが、見た目は、Sun の Java API と同

じであると。だからプログラマーが（9 ページの）一番左端の Method Call をするときに、今は見えませんが右端のところに Android の仮想マシンがあったら、上の宣言コードを一緒にそろえてあげれば、その下の実装コードは異なっていても、ちゃんときれいに動くと。これは野山さんに確認しないといけませんが、多分そのために宣言コードだけを利用していて、宣言コードの部分はかなりシンプルで分量的にも実装コードよりは少ないことが多いということであれば、分量の話などはうまくクリアできるのかなと。モジュール丸ごとというのは、恐らく結論としては逆になるのではないかと思います。

すみません、野山さん、宣言コードの分量的なところは、そういう理解でよろしいでしょうか。

【野山】　はい、一般的にはその理解で大丈夫だと思います。宣言コードのところは、ほとんどが 1 行ぐらいで呼び出し方を書いていて、インプリメンテーションのところはロジック部分で数十行とか数百行にわたるような形が一般的で、分量的には御認識の通りだと思います。

【伊藤】　ありがとうございます。

【山神】　ありがとうございます。

宮下委員、御発言をお願いいたします。

【宮下】　どうもありがとうございました。

まず 30 条の 4 のところで、プログラムの著作物については適用できないという分析というか御意見に関して、プログラムの著作物は、指令の組合せとして表現されたものであって、宣言コードは、モジュールの名前とか、特性を示す部分でしかなく、指令を組み合わせたものではないのではないかというのが、第 1 点の質問です。

もう 1 点、引用のところですが、通常、引用と言われたときに思い浮かぶ状況とは、表現された意図なり趣旨なりを、表現を受ける側が理解するために参考になるようなものを表現の受け手に対して示すということです。今回のような宣言コードを使うという文脈ですと、コンピュータの内部処理で便宜的に用いられるもので、必ずしもプログラマーに対する説明という意味というわけでも必ずしもないような気もするので、言葉としての「引用」とは、今一つしっくりこないなと。確かに引用と言われれば引用なのかなと思いつつも、やはり 32 条の、少なくとも立法者が意図したような状況とは違うのかなと思ったところです。その辺りについて御意見いただければと思います。

【伊藤】　後者の点から先にお答えしますと、委員のおっしゃるとおり、あえて違和感と申し上げますが、私も違和感を持っていたので、引用というのはちょっとないかな、というところから入っています。

（スライド 8）「報道、批評、研究その他」と列挙される中でどこにも当てはまらなさそうですし、私もちょっと無理かなと思っていたものの、実際、裁判例などで挙げている基準や要件に照らすと意外にいけてしまうかもしれないから、ちょっとこれはやってみようと思って今日に至っています。

引用が元々意図していたものや趣旨であるところの、引用することで著作物に触れる人に何らかまた伝えていくという、本来の引用とは明らかに違うなという違和感は私も持っていますが、実際に当てはめると、意外にちょっと切りにくいかもしれないというのが現時点のところです。

（スライド 6）もう 1 つの 30 条の 4 の、宣言コードは指令かどうかという話になってきますと、プログラムの著作物の著作物性の話と近接してくるかなと思っています。今日の私のお題としては、抜き出してくる部分が著作物であるという前提で検討するということなので、抜き出したものが指令ではないとすると、プログラムの著作物ではないという話もあり得るということと、私が実際に手がけた裁判例の中で、コメント部分が共通するというところをもって創作性や著作権侵害の話をしていた中の判決の一節の中に、「コメントはコンピュータに対する指令ではないのだから。」ということで、そこの部分ばさっと切られていた。もしかしたらせいぜい依拠性を基礎付ける部分なのかもしれないのですが、コメント部分が共通しているだけでは指令には当たらないと、ばさっと切った例があったのを今思い出しました。

そういう意味では、指令ではないと言われれば、確かにそもそも 30 条の 4 の検討にも入らないと思いますが、ただ一方で、やはり宣言コードも、このような名称でどういう型のデータを受渡ししますよということを説明してコンピュータに覚えさせるという意味では、これも一種の指令なのではないかというのが私の意見です。

【宮下】　ありがとうございます。

【山神】　奥邨委員が手を挙げておられますが、今の点少しだけ割り込ませていただいて、今の引用の目的の方は文言の末尾に「その他」と書いてあって、例えば絵画鑑定書事件でも、もう亡くなられた画家の方が確か「花」という題で多くの絵を描かれていて、どの花かよく分からないが鑑定書を書くには特定しないといけないという目的もあったということがあって、多分、伊藤委員の方向性は必ずしも間違いではないのかなという気がいたしました。

もう1つ、プログラムなのかという話は、正にIBF ファイルもそうですけれども、今回の宣言コードの部分のところも、実際、その後の実装コードとペアとなって、機械語にコンパイルしていった先では、例えばメモリの確保であるといったようなコンピュータに対する指示には、明確に、なっていると思います。多分そういう理解で間違いないかと思いますが、野山さん、いかがでしょう。実際にあのようなソースコードがどういう機械語に置き換えられていくのかという先まで見据えると、あそこはコンパイルすると消えてしまう単なるプログラムのコメント部分ではなく、ちゃんとコードに展開される部分かと思いますが、間違いないでしょうか。

【野山】　はい、そこも御認識のとおりです。

【山神】　ありがとうございます。すみません、割り込んでしまいました。奥邨委員、御発言をお願いいたします。

【奥邨】　今回引用でいけるのではないかというふうに伊藤委員に言っていただいて、今私、非常に心強く思っております。

私自身、NBL に論文を書かせていただいたのですが、このときに一応、引用でいけるというふうに方向性を示させていただいたのですが、ほとんど誰も賛同していただくことができず、無理なのかなというふうに思っておりましたが、今日は伊藤委員からも、いけるのではないかというふうに言っていただいて。

基本的に考え方は一緒なのですが、ただ、例のパロディモンタージュの件は、先ほど岩原委員からもあったように、分量とか主従関係だけでいってしまうと、コンピュータプログラムのサブルーチンを使うのも全部 OK ではないかというふうに取られてしまう可能性があるので、やはりあれは用いない方が良いと思うのですね。

そもそも引用も、条文と一致していないではないかとよく言われますが、「引用の目的上正当な範囲」というのが飛んでしまっているわけですよね。パロディモンタージュ等々は、ある程度引用の目的が前提にあった上での話をしていますが、今回は、今回の引用の目的を特定しないといけなくて、そうすると、全く同じ機能を享受するという形で使うことが引用の目的と言えるのかという問題の立て方があり、今回だと、例えばアメリカの transformative と全く同じ議論をすれば良さそうな気はしています。

ですから、宣言コードを使うというのは全く同じ機能を使うというわけではないのだと。本来はプログラムの再実装という目的のためにやるのであって、その再実装という目的のために必要な範囲かどうかということで捉えればいいのかなと思います。

そうすると基本的には条文に忠実に要件を当てはめていくということだと、余地はあるのかなと思います。

もう一つ、今、宮下委員がおっしゃった点、引用という言葉の一般的な用語とちょっと違和感があるというところです。

まず引用という言葉自体は、辞書で引くと、自分のものの中に他人のものを持ち込むこと、がまず出発点なので、これは満たすのですね。問題は、著作権法上の引用かどうかということなのですが、そのときに宮下委員がおっしゃった例は、これは、それこそ人間が思想・感情を享受するタイプの引用をおっしゃっています。今回は、プログラムの機能を享受する話なので、機能を一部使わせてもらうというのは、そういう意味の引用としてあり得るというふうに思います。ここは既にリバースエンジニアリングの議論で表現の享受と機能の享受と分けましたが、これがそのまま適用できて、今回は機能の享受ということなので、そういう意味では機能の面の引用というのがあり得るという論は立てられる余地があって、ほかと余り矛盾しないと私自身は思っています。

ある意味で、特に絵画鑑定証事件などが出してきている総合考慮、絵画鑑定書事件自体は非常に分かりづらいですが、総合考慮の余地というのは、基本的には transformative な場合のフェアユースの要素と似ているところがありますので、うまく当てはめればいけるのではないかなと個人的には思っております。

【宮下】　ありがとうございます。
　引用の点、私も機能を呼び出すという意味での引用は確かにあり得ると思いますが、先ほど申し上げたかったのは、32 条で想定されているような引用では恐らくないだろうなということでしかないので、その点、一応補足しておきます。
　それから、30 条の 4 のプログラムの例外規定のところですけれども、あそこで例外として除いているのはプログラム著作物の実行なのですよね。宣言コードだけでは実行はできないと思うので、そういう意味からしても、30 条の 4 の適用除外の規定が適用されないという考え方もあり得るのではないかと思ったのですが、この点はいかがでしょうか。
【伊藤】　非常に難しくて、考えがまとまらないのですが、今の御質問は、30 条の 4 の第 3 号に「プログラムの著作物にあつては、当該著作物の電子計算機における実行を除く。」とあり、除かれているのではないかという御指摘でしょうか。
【宮下】　そうです。ここで除いているのは、プログラム著作物の実行なので、仮に宣言コードがプログラム著作物の一部分だとしても、それを実行しているわけではないですから、30 条の 4 の規定が適用されないことには必ずしもならないのではないかという趣旨です。
【伊藤】　「実行」をどう捉えるかということがあると思いますが、宣言部分を読んで、何らかコンピュータに処理が動く、呼ばれたことが分かるということ自体も、実行の一部なのではないかと私は捉えますが、ここも先ほどの指令なのかどうかの話と議論が似てくるでしょうか。
【奥邨】　今の「実行を除く。」ということは、実行自体が柱書の「享受」に当たるという理解ですよね。
【宮下】　そうだと思います。
【奥邨】　そうですよね。ですから、実行を除いても享受に当たるというのは別途議論があるので、3 号に当てはまらないということで 30 条の 4 が適用されないということには必ずしもならない。飽くまで例示なので、柱書の方を素直に読めばよく、3 号は余り問題にならないのではと思いますが、いかがでしょうか。
【宮下】　なるほど。そう考えた方がすっきりします。ありがとうございます。

５　第５回委員会−

(1) サマリー

第 5 回委員会（2023 年 3 月 16 日開催）では、第 4 回委員会のうち特に伊藤委員の報告に係る「日本の著作権法のもとでの権利制限規定の適用可能性」に関して、委員間で更なる質疑応答及び討議が行われた。

(2) ディスカッション

【山神】　今回は、前回（第 4 回）の資料 3 に基づく伊藤委員の御報告についての落ち穂拾い的な議論をして、それからディスカッションをして、その後に作成中の報告書についてざっくりとしたアウトラインを検討するという流れで、議事を進めてまいりたいと思います。

それでは、前回の続きについて自由に発言していただくということで、どなたでも結構ですので、御意見を頂戴できればと思います。いかがでしょうか。

伊藤委員、前回話しそびれたことなどあればお話しいただければと思いますが。

【伊藤】　特に言い残したことがあるということはございません。前回、権利制限規定との関係の話をさせていただいて、その議論の振り返りということですけれども、日本法の権利制限規定の下で、Google Oracle 事件と同様の事件があったときには、32 条の引用の適用の可能性があるかということについて検討して発表したところ、当然私もそちらを拝見しておりましたが、奥邨先生もかつて NBL の御論稿[9]で同じようなことを御発表されたというような話がありました。

前回の議論の振り返りということで発言させていただきました。

【山神】　ありがとうございます。

私から伊藤先生にお伺いしたいのは、多分、権利制限規定というのが一つの逃げ方というか、妥当な解決を導く方法としてあり得るかと思うのですけれども、それ以外に、著作権法 10 条 3 項の議論です。あれは著作物性がそもそもないものを確認的に定めているものだと思いますけれども、そういったものを使う可能性については、先生としてはどういうふうにお考えでしょうか。

【伊藤】　ありがとうございます。実は、この判決が出された直後に、SOFTIC が開催したセミナーがありまして、そのときの資料でも同じように 10 条 3 項各号の該当性を検討したやに記憶しております。資料を探しますので、少々お待ちください。

【山神】　お探しの間つなぎますと、個人的には多分 2 号（規約）なのかなと思っていまして。2 号については、記録がないだけなのかよく分からないのですけれども、立法当時、そこまで詳しく突っ込んで立法されてはいなかったようなのですが、一応、プログラムを著作物として保護するということを考えたときにいろいろな手当が必要ですということで、プログラムの著作物の周辺情報といいますか、そういったものの著作物性について、こういったものは保護から外しましょうと。

例えばプログラム言語であれば、Java というプログラム言語を Sun Microsystems が作って、それが従来のプログラム言語よりもいろいろと使いやすいということでどんどん流通していったわけですけれども、そのときに、同じ Java という言語を作る場合で、こうした話は昔からあったわけです。プログラム言語と処理系といった使い分けがされていて、X 言語というのがあったときに、それを実際にコンパイルなどするものを処理系というふうに言うのですが、A 社が提供するコンパイラ、つまり（X 原語という）プログラム言語をいろいろ処理してプログラムを作っていくための様々な一連のツールを例えば A 社が音頭を取って作ったところ、わりとお金になりそうだということで、それに対して A 社と対立関係にある B 社も X 言語を処理できる同じような処理系を投入したところ、それが著作権侵害になるのかといったとき、プログラム言語の部分については保護が及びません、ということになる。

それに対して、規約というのは、プロトコルと先ほど申し上げましたが、プログラムを扱っているときにいろいろな取決めが出てくる。例えばデータのやり取りの仕方であったり、それから、

[9] 奥邨弘司「Google v. Oracle 事件合衆国最高裁判決−Java API を実現するプログラムのフェア・ユースについて」NBL No. 1202（2021 年 9 月）。

API と呼ばれるものでありますけれども、実際に入り口の部分について、プログラムのモジュールはこのように動くのだというふうなところがいわゆる宣言コードというところで、それについては、恐らく規約、「特別の約束」に相当するのではないかという話です。

ですので、恐らく、プログラム言語、規約、解法といったときに、解法はアルゴリズムですが、プログラム言語と解法は、今回の事件に関しては、厳しいのかなと。一番の落とし所は規約なのかなと思っているのですけれども。

すみません、長々とお話をいたしましたけれども。

【伊藤】　今、2021 年 5 月のセミナーの資料をお示ししました。

（セミナースライド 6、7）今先生がおっしゃったとおり、「規約」への該当性はそれなりに高いと私も思っておりまして、たまたま中山先生のソフトウェアの法的保護（中山信弘『（新版）ソフトウェアの法的保護』）にも、インターフェースなどがこれに該当するとされています。ただ、インターフェースがプログラムの形で記述されている場合には、「事実上デッド・コピーのみが侵害となろう」と。保護の可能性もあり得るというようなことが検討されていて、今回の Oracle のケースでいいますと、部分的にはインターフェースというか、正に API のところがほぼデッドコピーされているので、これをそのまま読むと、一応、インターフェース自身がプログラムの形で記述はされているし、それ自体がデッドコピーされているので、侵害になってしまうという評価もあり得るのかなと思いながら、問題提起だけをしたというのが、セミナーのときの状況です。

ただ、一方で、今山神先生のおっしゃったことは、私もほぼ同意をしていまして、結構、2 号の規約に当たり得るのかなというふうにも思っております。

【山神】　そこの読み方なのですけれども、インターフェースがプログラムとして実現されていたときに、今回は宣言コードとそれを実際に実体化する部分が常に分かれているわけですけれども、それを全部デッドコピーしてしまうというのはアウト、そこまでは保護範囲にギリギリ入れても良いのではないかという議論なのかなと私は理解していました。

つまり、うまくきれいに切り分けると、恐らく Google も、ここまではセーフでここから先はダメなのだろうなということで、ダメなところを自分で作ったという考え方だったと思います。中山先生のこの記述を踏まえても、ギリギリいけそうかなと思うのですけれども、いかがでしょうか。

平嶋先生、どうぞ。

【平嶋】　自分も検討したときに考えたのですが、多分「規約」に読めるかどうか、ギリギリのところかなと思っています。確かに API に関しては、それを実装するコードの部分は Google が全部書いているのですけれども、結局それを呼び出すときにはどうしても API の宣言コードと呼ばれている領域を使うことになって、通常 Java でプログラムを組む方は恐らくほとんどそれを使っているという状況を考えると、限りなく規約なのですが、もともとは Java の言語として、API は必ずしも完全に組み込まれているというほどのものではなく、その下のもので組むという前提だったのでしょうが、API をほとんど簡便化のために使っているという前提ですと、多分規約の概念が、プログラム言語の最初の体系からいろいろ付け足しで使われていくというプログラムの慣習みたいなものによって変わっていっても良いのかなと思ったりもします。そうすると、確かに規約と読んで、それで著作物性のところからクリーンにするという考え方もあるかなと思っていたところではあります。

3 回目で報告をした際に、著作物性がギリギリあってもおかしくないのではないかといった話とは少し違うのですが、逆に規約と読めるのであれば、著作権の行使という話に余り踏み込まないで済むという意味では、規約の概念の解釈で柔軟に読むという考え方はあるかなと思います。

規約については、最初に言語として組まれているものから派生してさほど広げられないという発想でいくと、それはもう権利行使のところで調整せざるを得ないと考えますが、規約の概念をもう少し柔軟に、プログラマが慣行で API にプラスして決まった定型的な処理をパッケージにして、それを呼び出す機能の表現があって、それも規約に読み込むということで、広く著作物性の評価として著作権の保護から落としていくという発想はあるのかなと思いました。

【山神】　ありがとうございます。ほかにいかがでしょうか。

奥邨先生、お願いいたします。

【奥邨】　ここのところ、今回は、例えば今の規約の話ですと、10 条 3 項を見ると、規約というのはプログラムそのものではないのですよね。プログラムとは別の存在です。

今回複製されたのはプログラムなのですね。宣言コードの部分ですが、これもプログラムであることは事実だと思います。Java を実現するプログラムの中の宣言コード部分と実装コード部分があって、宣言コード部分をコピーしたのですが、この宣言コードというのは正にプログラムだと。それで、概念としては、規約というのはそれとは別に存在しないといけないものであって、その規約とマージしているのが、宣言コードだったということなのではないかと思っています。

規約としては、どちらかと言うと、技術の方から御紹介いただいたような、本になっているような形の、あの中に書かれている抽象的な決め事というのが規約であって、それをプログラムの形にしたということ。表現形式は違っているのだけれども、マージしてしまって、一対一でしか存在し得ないというのが、Java の呼び出し方の部分だということだとすると、これは形式としては著作物になり得るのだけれども、マージしているので創作性がない、若しくは表現と呼べない等の理由から、保護されないという形になるのではないか。私自身はそういう整理かなと思っています。

逆に言うと、ここのところでいろいろとバリエーションを取ることができたというのであれば、規約を実装したとしても、仕組み上バリエーションがあっても規約の実装ができるというのであれば、マージしていないということで、プログラムの著作物として創作性があると言われた可能性があると思います。ただ、Java の仕組み上、それはあり得なかったということなのかなという理解をしております。

【山神】　おっしゃるとおりですね。プログラムの中でも、例えばどこまでがコードでどこまでがデータなのかという議論もありますけれども、正に今奥邨先生が御指摘になったように、規約を実現せざるを得ない部分がある。Java の仮想マシンのサブセットというのでしょうか、問題となった Dalvik と呼ばれる仮想マシンがあって、それを Google が作りましたと。本来は、宣言コードを流用しているというのは絶対に分からないはずなのですね。分からないけれども、オープンソースだったので、全部、どうやって使っているかというのがつまびらかにされて、そうすると確かにプログラムの中の、データというよりどちらかというとプログラム寄りの部分があった。そして、そこは、前回までの議論で、他に書きようがないのだと。それを変えてしまうと、Dalvik という仮想マシンに対して、Java のほかの仮想マシン用に書いたプログラムを投げても、ちゃんと動かない。エラーが起きてしまう。だからちゃんと動かすためには、それがそろっていないといけない、ということが多分マージということになるのでしょうかね。

いろいろ整理していただきまして、私も大変クリアになりました。ありがとうございます。

いかがでしょうか。すみません、いきなり振って恐縮ですが、上野先生、いかがでしょうか。せっかくですのでコメントがありましたらお願いいたします。

【上野】　ありがとうございます。今の議論ですと、規約か、あるいは規約を表現したプログラムに創作性がない、あるいはマージしているから著作物性がないという方向性だと思うのですが、まず奥邨先生は、プログラムではあるとおっしゃったと思います。プログラムに当たるというためには、著作権法上、「電子計算機を機能させて一の結果を得ることができるようにこれに対する指令を組み合わせたもの」ということですから、指令の組合せとして電子計算機を機能させるものである必要があるのでしょうね。

ですから、今の議論で、プログラムなのか規約なのかが分けられているというふうに理解するならば、今回の宣言コードというものがプログラムなのか、それともプログラムではないのか、という問題になると思うのですが、そこでは一の結果を得ることができるか、コンピュータを機能させるのか、指令の組合せと言えるかどうかによるのだろうと思いますけれども、これはもう少し更に御意見を伺いたいところであります。

プログラムと本当に言えるのかなと思ったりはするところですが、プログラムと言えるとするならば、それを正確に表現すると、プログラムなのだけれども、規約から導かれる、規約をもとに表現するならば、そうならざるを得ないみたいなところがあるので、アイデアとマージしているから、著作物性がない。日本法で言うならば、あるいはマージ理論を取らないで説明するなら

ば、創作性がないという説明になるのかなというふうに思ったのですが、その前段の部分をもう少し御意見を伺うことができればと思った次第です。

【山神】　ありがとうございます。今の点、いかがでしょうか。

今の上野先生の御発言に乗っかるとすると、著作権法 10 条 3 項 2 号の文言ですね。「特定のプログラムにおける前号のプログラム言語の用法についての特別の約束」というのが、一体何を考えていたのか。良い立法資料などがあると良いのですが、正にこの文言自体が、実は今上野先生が御指摘されたように、規約という取決めがあって、それが実際にプログラムの中に出てきて、奥邨先生のおっしゃるように、マージをせざるを得ないようなものなのだと。

私が授業で規約の説明をするときによく引き合いに出すのが、今はほとんど実用性がなくなりましたが、一昔前ガラケーというのがありまして、ガラケーから電話番号を引っ張り出すのが結構大変で、機種変更したら前の情報をどうやって移すのかということを考えたときに、専用の充電ケーブルには実は信号が乗るようになっていて、それをパソコンと接続して、そこから携帯電話の電話番号をバックアップする、というふうなソフトがございました。

これはオープンな仕組みではないのですけれども、ソフトウェア開発会社が頑張っていろいろテストをしながら、ドコモさんのこの機種だとこんな感じで信号が流れるということで、プログラムを書く。実際にはそういうことはなかったわけですが、例えばドコモさんが純正で作っている電話番号のバックアッププログラムがあったとして、サードパーティの会社が独自に、ケーブルを流れる情報をチェックして、広い意味でのリバースエンジニアリングをして、こういうふうにデータが流れてきて、例えば電話番号帳の氏名であるとか電話番号であるとか、そういった吸い出しのときのデータのフォーマットが分かって、それをそのままコーディングすると、ドコモさんの純正のプログラムととても似てしまう。これが著作権の侵害になるのかというとそうではなくて、それはその正に「特別の約束」なのだと。実際に「プロトコル」という用語は、比較的、そういう信号のやり取りをするところで使われるわけでありますけれども、そうすると、もしかしたら 3 項の 2 号はそういうもの、正にこの Google v. Oracle 事件のために用意されていた規定なのかな、などと思ったりもしました。

すみません、勝手に発言しましたが、この辺り、例えば奥邨先生、いかがでございましょうか。

【奥邨】　ちょっと画面共有させていただきます。

（第 3 回資料 1 スライド 43）先ほど上野先生がおっしゃったところとの関係でいくと、まず、ソースコード上は、確かに宣言コード部分だけ見ると、上野先生おっしゃったように、指令の組合せになっているのかという問題がありそうなのですけれども、ただ実際は、宣言コードと実装コードが別々に存在するのではなくて、こういうふうな形で混然一体として宣言コードと実装コードが Java を実現するプログラムとして存在しています。「プログラム」というのは「プログラムの著作物」という意味ではなくて、技術の用語でいう「プログラム」として存在している。

その中の黄色い部分を宣言コードと呼んでいるということになると、これは、全体として見れば、一の結果を得るための指令の組合せになっていて、宣言コードというのはその指令の組合せの中の一部分だけを色付けて呼んでいるということだけだと思うのですね。通常のプログラムでも、例えばヘッダの部分のような部分は多数あるのですが、そこだけ取り出して「プログラムではない。」という言い方をするかどうかというと、先ほど山神先生がおっしゃった、どこがデータでどこがプログラムか、という議論があるのと同じように、溶け込んでしまっている場合は全体として見て良いのではないか。そうではなく、細かく分解していってしまうと、どこもプログラムではなくなるということになると思いますので。ですので、今回はそういう意味で、私はプログラムかなというふうに思っています。

ただ、API を実現するプログラムの中で、先ほどありました、API の決まり事をコーディングするのにはこれしか方法がない、という部分が黄色いところで、実は、多分技術の方は宣言コードとか実装コードとかいった呼び方はしていないのではないかと思うのですね。多分これは訴訟の中で説明がしやすい形でいろいろと出てきているような気がしていまして、ふだんは余りそのことは気にしていないはずではないのかなと思うのですね。

ですから、黄色く色付けされた部分は、飽くまでプログラムの中の、API に合わせるとそのようにしか書きようがない部分という趣旨なのかなと理解しています。

もう一つ申し上げると、あとは規約かどうかというところですが、個人的には、余りこだわる

必要はないのですけれども、もしかすると、規約というよりもプログラム言語で良いのではないかと。そもそも Java API というのは、Java の言語の体系の一部と考えれば良くて、正にこれ自体が Java 言語というプログラム言語自体を動かすプログラムなので、そういう意味では、もともと 1 号か 2 号か 3 号かでそれほど区別する価値はないのですが、あえて言うと、先ほど山神先生がおっしゃったように、どちらかと言うと、「特定の」と付くので、いろいろなものに適用できるものはニュアンス的にどうなのだろうという御指摘もありましたが、それからいくとプログラム言語の体系の一部みたいなところに取っても良い。ただ、やはり今回は、一般で言うところの API とは少し違うのではないか。普通の、ウェブの API を公開している、という話と少し違って考えた方が良くて、Java という言語を作る API、言語の一部の API ということなので、どちらかというと言語の一部だというふうに理解した方が、もともと考えていないものなのでどれに区別するか非常に難しいのですけれども、あえて区別すると、言語寄りかなと、個人的には思っています。

そもそも Java の関数というのは、オブジェクト指向だとだいたいそうですが、どちらかというと命令語を増やすような意味があります。そういう意味でも、言語の一部と考えた方が良いのかなと個人的には思っています。余り区別の実益はないのかもしれませんけれども。

【山神】　ありがとうございます。なるほど、そうすると 1 号も検討した方が良いということはあるかもしれません。

先ほど奥邨先生がシェアしてくださった、黄色や青で塗り分けられたところは、例えば昔、プログラムを著作物として保護するかしないかというところで、無理やり著作権法に入れた一つの理由が、一応言語っぽいもので書いていると。直ちに英語とか日本語といったものとはつながりませんが、そこにある程度類似した体系を持ったもので書かれているからだと。そうすると、翻って考えると、あれが小説であったとしても、確かに全体としては小説なのだと。

ですから今回問題となった宣言コードも、大きなくくりでいえば小説であって、その小説の文言の中で、一部は著作物がない部分なのだというふうに考えると、普通の言語の著作物、典型的な小説などでもそういうことはある。ものすごく小さく切り刻んでいくと、例えば「雪国」の冒頭シーンなど、ある程度長くなってくると著作物性が出てくるのかという話はあると思いますけれども、「国境の」、「トンネル」、「しかし」とか、正に今回の宣言コードはテンプレート的なところがあるわけですが、テンプレートとして使われるような記載というのが、小説の中に含まれているというのはよくあることですよね。そうすると、何となく、10 条 3 項を持ち出すというのも一つですけれども、それも捨て去ってしまって、一般的な言語の著作物的な観点から、その中の著作物性のない部分というのがあって、という考え方ができる。そうすると制限規定などを無理やり作るのではなく、著作物性がないと正面から説明してしまった方が良いような気がいたしました。

上野先生、どうぞよろしくお願いいたします。

【上野】　とても勉強になりました。ありがとうございます。ここまでプログラムについて深く議論することはなかなかないと思います。

確かに、全体としてプログラムであれば、その一部だけ取ってもプログラムでないものであっても、プログラムの一部ということなのかもしれないですよね。なるほど、宣言コード部分もプログラムの一部だった以上はプログラムだということなのかもしれませんが、それはプログラムであるかどうかによって、著作物性の認定が変わらないのであれば、別にプログラムであるかどうかとか、規約であるかどうかというのは関係がないかと思うのですけれども、10 条 3 項を創設規定と読むかどうかで、もしかしたら結論は変わってくるのかもしれませんよね。

つまり、プログラムの一部だとか、あるいは規約の一部だといっても、結局は創作的表現であるかどうかだけが問題になるのだというのであれば、ある一部がプログラムかどうかといったことは余り議論する必要はなのかもしれないのですけれども、10 条 3 項は創設規定であると。もしかしたら創作的表現に当たるようなものであったとしても、それは積極的に、創設規定的に著作物性を否定するという規定なのだと読むのであれば、やはりそれはどちらに位置付けるのかということが、意味を持ってくるように思いました。

ですから、先生方の議論、非常に勉強させていただいたのですが、その背景に、10 条 3 項をどういう性質だと捉えているのかということが問題となるものなのかなというふうに感じた次第です。

【山神】　ありがとうございます。ほかにいかがでしょうか。
　平嶋先生、お願いいたします。

【平嶋】　上野先生が今おっしゃったようなお話で、多分、先ほどの API の「java.lang…」などというのは、私も第 3 回の報告で少し申し上げましたように、手続の流れを表現としてどういう言葉でまとめるか、キャッチフレーズのような形で「lang.Math」といった呼び方を付けていて、自由度としては面倒くさい名前を付けようと思えば幾らでも付けられるけれども、API の宣言の呼び名として余り不自然なものを付けるのはおかしいので、表記の仕方は自然と「lang.Math」などというようにある程度固まってしまうところがある。ネーミングとしてはいろいろな付け方があり得なくもないのではないかというところで、低いけれども創作性ゼロとは言えないのではないかという話かと思いますけれども、そこは先ほどの、規約かプログラムか解法かのいずれかでバサッと切ってしまうやり方もありますが、仮に何らかプログラムの著作物であるという前提を満たしたとしても、恐らく創作性は限りなく低いと考えてしまって、著作物性が切れてしまうとしてそこで落としてしまうとか、あるいは、保護の対象としても、著作物性が低いということなので、権利の範囲が相当狭くなるということで調整していくことになるのかなと思います。
　ですので、上野先生が先ほどおっしゃったような形で、3 項に入るかどうかというところでみていっても、あるいは仮に入らないとみても、創作性のところで結局ある程度制約はあり、創作性を付加するという部分でもないのではないかというところで見ていくと、結果的には著作物性が限りなく低くなっていって、という整理になるのかなと思いました。
　ですので、先ほど奥邨先生がおっしゃった、解法かプログラム言語かということは、私は、これだけプログラマーの中で定番化された使い方ということなので、解法という言い方もありかなとも思ったのですが、プログラム言語とみるか、規約とみるか、プログラム言語そのものだという言い方でも、実はそれほど無理筋ではないのかなとは思っております。

【山神】　ありがとうございます。
　今、解法という話がありましたが、例えばソーティングをするときにはいろいろなソートの仕組みがあります。ソートですとかなり議論は出尽くしているので余り新規のアルゴリズムはないかと思いますが、どういうふうに計算するのかという辺りが技術の中心となるときにはソフトウェア関連発明としてどうやって保護するのかという議論があったかと思います。今回の API の宣言コード部分というのは、解法だとするならば、その解法の外側の皮だけの部分で、解法は卵の黄身の部分みたいな感じだと思いますが、そういう理解で間違いないでしょうか。趣旨を平嶋先生にお尋ねしたいと思いますけれども。

【平嶋】　そうですね。結局プログラムを組んでいくときに宣言コードで普通にメソッドをまとまりで呼び出すときの取っ掛かりという位置付けとして使われているところが問題になっていると思います。そういう意味で、広い意味でそれは解法の一種といいますか、これはもう定番で、クラスを呼び出して処理させる、そこに組み込む表記という位置付けに過ぎないので、それを読み出せば後は自然と実装コードが組み込まれて処理されるという流れでいくと、そこはむしろ解法の取っ掛かりとみてしまえば良いのではないか、そういう考えもできなくないのかなとは考えていました。
　結局、実装コードの部分が実際の処理にかかってくるわけなのですけれども、そこは一般化されている部分なので、その実体の部分は広い意味で一つの解法として処理されていくと考えると、呼出しに使われる部分、ここでいう宣言コードと呼ばれている領域は、広い意味で、処理の一つのまとまりという分け方でもそんなに無理はないという考え方ができると思っていたのですが。

【山神】　例えば先ほどのソーティングで、我々が定期試験を採点するときに、バラバラに返ってきた答案を番号順に並べ替えますよね。例えばデータが 1,000 個与えられて、それが配列に格納されていて、それをある API に渡すと、100 個とか 1000 個を一瞬で並べ替えて、その結果をまた同じ配列に戻すというプログラムを考えたときに、API としては、多分、ヒープソートとかバブルソートとか、異なる解法、どのようなアルゴリズムを使うかはプログラムする人の自由かと思いますけれども、外面はそろえることができるかと思うのですね。
　そのとき、解法というのは、恐らく、バブルソートにするのか、ヒープソートにするのかといった、ソートのアルゴリズムなのかなと。それを API を使う側から、どのようなアルゴリズムで

やっていただいても結構、なるべく早く結果が出れば良い、100 個くらいだとどのアルゴリズムを使っても余り変わらないけれども、1,000 個とか 1 万個とか、数が増えていくにつれて計算量が違ってくるというところがソートのアルゴリズムを選択するときの肝で、スピードとメモリの使用量などで使い分けるかと思いますが、そのときに、今回の API というのは、これだけのデータを渡してこういうふうに結果を返してくださいというところだけを書いているので、少し違うのかなと思ったのですが、そういう理解だとおかしいですかね。

【平嶋】　ありがとうございます。そういう意味では、むしろその処理というものが一つのパッケージになっているというイメージがあります。処理を呼び出すために Java で書いていくとき、定型的な処理を呼び出すときの記述という領域が宣言コードという位置付けになるのかなと。ですからそこは一体化されているということだと思います。宣言コードというところと特定の一つの処理の体系が完全にひもづけられている。恐らく、そういう前提で宣言コード部分を捉えたときに、それはもう具体的な処理とほぼひもづけられているのではないかという捉え方ができなくもないかなと。

そうすると、それは規約とほぼ同じと言うといけないのでしょうけれども、ある意味ひも付けができてしまっているという理解ができるのかなと思ってはいました。

それを結局言語でやっていくのでしょうということであれば、プログラム言語の一つの体系だ、言語体系そのものを成すものだと理解してしまう、先ほど奥邨先生がおっしゃったように、言語体系の方に持っていってしまうやり方もあるのかもしれません。

【山神】　ありがとうございます。

宮下先生、どうぞお願いいたします。

【宮下】　Declaring Code についての私の理解が十分ではないのかもしれないのですが、Declaring Code は、関数とか、その関数で処理するデータの種類とか、アウトプットされるデータの種類であるとか、そういうものを定めたものというふうに理解していました。

それで、コンピュータに対する指令自体は Implementing Code によって行われると。これは、名前とどういう処理をするのかということを定めた、いわば特定のための符号でしかなくて、それ自体はコンピュータ、電子計算機に対する指令ではないのではないかと思っていたのですね。

そうだとすると、個々の Declaring Code はプログラム著作物で言うところの指令ではないと。だとしても、先ほど奥邨先生が御指摘されたとおり、Implementing Code と組み合わされた全体の明らかなプログラムの一部であるという意味でプログラム著作物というふうにみるのか、あるいは Declaring Code はそもそもコンピュータに対する指令が含まれていないので、これはコンピュータプログラムではないと。コンピュータプログラムでないとすると解法であるか、言語であるか、あるいは規約であるかは余り問題にならず、Declaring Code という素材をどう選択して、どう組み合わせたのとかという意味での、編集著作物としての著作物性が問題になるにすぎないということなのではないかと思っていたのですけれども、そういう考え方はいかがでしょうか。

【山神】　ありがとうございます。多分、コンピュータに対する指令であるかないかということについては、やはり指令ではあるのだと思います。どういうことかというと、例えば「int」とか、変数が文字列なのか、浮動小数点なのか、整数型なのかということが宣言コードの中に含まれていたりするわけですが、では、それがバイナリで最終的にどうなるかというと、やはりそれぞれ違ったコードが作り出されるわけなのですね。つまり、それによってプログラムが最終的にオブジェクトコードになったところで、どういうふうにレジスタに配置していくとか、どの辺りにメモリを確保していくのか、というところが違ってくるので、そこはやはり指令と言わざるを得ないのだろうという気がしています。

それが指令ではないとなると、宮下先生のお話に流れていくのだと思うのですけれども、なかなかそこは苦しいのかなという気がしています。

ただ、最近のコンパイラは結構賢くて、ソースコード全体を読んで最適化という作業をしていくのですけれども、そのときにも宣言コードのところなども含めて全体を読んで最終的なコードを出してくれますので、それが多分、いわゆるソースコードとオブジェクトコードの関係という観点では、指令なのだろうという気がいたします。

宮下先生、いかがでしょうか。

【宮下】　すみません、具体的にどういう形でバイナリコードに変換されるのかよく分からなかったので、私の理解は Implementing Code がコンピュータに対する指令として、それがソースコードになって、それをコンパイラなりでバイナリコードに変換するので、Declaration Code 自体はコンパイラでバイナリコードに変換するときに読み込まれるわけではないのではないかと思っていたのですが、そうではないのでしょうか。

【山神】　読み込まれます。正確には、今回の場合は Java の実行プログラムが問題となっているのではありません。我々のスマートフォンで言えば、スマートフォンにはいろいろなアプリがありますね。あれは Java 類似のコードでプログラミングされて、携帯に転送すると動くと良いな、と。ちゃんと動くのはそれをちゃんと解釈してくれる仮想マシンというのがあって、Java の正式な仮想マシンとは違うけれども、Android のスマートフォンは全部これに統一していますと。言語体系も少しサブセットだけれどもほとんど同じですと。今は訴訟が起きたときの Dalvik という仮想マシンから更にバージョンアップした仮想マシンが使われているわけですけれども、仮想マシンという点では同じで、今回の宣言コード部分というのは仮想マシンを最終的にビルドするときに影響がある部分なのですよね。それは Java のアプリケーションプログラムを作るときにもコンセントの受け口と差し込み口みたいな感じで意識しないといけないのだけれども、今回の侵害訴訟という観点で言うならば、Sun Microsystems とか Oracle が作っている純正の仮想マシンがあって、それに対してサブセットであるけれども似た仮想マシンを Android 系のスマートフォンに組み込んで販売していた。それを作った経緯からするということで、その中のソースコードを見てみると、元々の API そっくりのところがあって、ということなので、読み込んで、Dalvik の仮想マシンというのが作り上げられてというのは間違いないというところですね。

【宮下】　Declaring Code の機能というか、ファンクションもよく理解できなかったので。単なる名称だとしたら指令ではないのだろうと思っていたということです。

【山神】　全体として読み込まれるので、どうしても最終的には影響があるわけですね。これはアプリでも同じで、宣言コードというものがあって、今回問題となった Java の API はこういうふうな書き方をしますということを踏まえて、携帯用アプリをお作りになった方がおられるとすると、どういうふうに書くかというと、API のインターフェースと合わせていろいろ宣言をしていく。そうすると、それは最終的にはアプリの中に何らかの形で反映されるということになります。したがって、先ほどのコンセントの差し込み口と受け口のどちらもそれぞれのパーツに対して影響を及ぼしているということになります。

よろしいでしょうか。ほかにいかがでしょうか。

せっかく伊藤先生の資料を出していただいているので、本来的な著作権法の制限規定、ほかに使えるものというのはありますでしょうか。資料の中でいろいろ御説明いただいたものは既に前回ある程度議論はしたかと思うのですけれどもいかがでしょうか。

いきなり振って恐縮ですけれども、椙山先生もプログラムの著作物について一言も二言もお持ちだと思いますので。

【椙山】　そうですね。山神先生が文学みたいな話とプログラムみたいな話を一緒にされているのは、少し僕には違和感がありました。特許に近いというわけではありませんが、技術的なものについて言うと、やはり文学のようにバリエーションがあるものとは違うのではないかと私は基本的に思っていて、そういう技術的なものについて余り強い保護をしてはいけないということで、著作権法にプログラムを取り込んだときにも 10 条 3 項みたいなことで調整しようとしたということだろうと思っているのですね。

そうだとすると、基本的な趣旨はどういうことかということで言えば、やはりプログラムというのは技術的なものだから、文学みたいなバリエーションがないという前提で、ちょうど良い保護にしましょうという話だろうと思うのですね。それで 10 条 3 項に 3 つくらい入れているのですけれども、僕は多分「言語」で良いのではないかという感じはしています。ですから、奥邨先生の考えと同じなのですけれども、そういうことだと思っていますので、それを 1 号、2 号、3 号に振り分けるということには、実質的には余り意味がないのかもしれないという感想を持ちました。

【山神】　ありがとうございます。私も決して小説と同じということではなくて、飽くまで今椙山先生がおっしゃってくださったとおりの理解であります。プログラムを書くときにはいろいろ制約がある。

今回の Google v. Oracle と少し違いますけれども、システムサイエンスなどもそうですよね。計測機器からデータを取って、それをプリントアウトするときにプリントアウトの紙切れをどうやって調べますかということをやったときに、それはそう書かざるを得ないでしょうというところがあって。

ですから、小説とプログラムというのは言語っぽいというところが似ているけれども、多分、技術的性格によって、著作物性がないというところが広がっているのでしょうね。

【椙山】　共通点は確かにあると思うのですよね。自然言語と人工言語みたいなところがあって、技術的かどうかという意味ではかなり違うのだけれども、一定の目的を持ってコミュニケーションするためのツールみたいな感じでできているわけですね。そして、一つの言語で多数の人が同じことをやろうとしているわけですよね。そうすると、誰が作ったにせよ、一旦多数の人のルール、共有物みたいな感じでみんなが使うようになったときには、やはり言い出した人の保護というのをそこまで認めると非常に具合が悪くなるというのが基本的な考え方なのではないかというふうに僕は思っています。

【山神】　おっしゃるとおりですね。ありがとうございます。

そうしましたら、また突然の御指名で恐縮ですけれども、谷川先生いかがでしょうか。谷川先生もこういう議論、お好きだと思いますので。

【谷川】　ありがとうございます。そうですね。10 条 3 項の規定が創設規定か確認規定かということですよね。やはりプログラムの中には、おっしゃるように、そもそも創作性がないもの、規約としてアイデアに属するものも多いと思いますので、僕は確認規定ということで理解をしていたので、10 条 3 項の規約かどうかという議論をしなくても、普通のアイデア表現二分論で良いのかなと思っていたのですけれども、そこは余り意見の一致はないということなのでしょうか、という疑問を持ちました。

【山神】　どうなのでしょうか。僕も確認規定なのかと思っていたのですが。先ほどの議論では、多分一致はしていたのかなと。すみません、私の理解が違っていたら、挙手をしていただければと思うのですけれども。いかがでしょうか。

【谷川】　そうであれば、10 条 3 項の 1 号か、2 号か、3 号か、という議論はしなくても良いかなと思って聞いていたのですけれども。以上、感想です。

【山神】　きれいに分ける必要はないのではないかという話は先ほどから度々出ておりましたね。ありがとうございます。

そうしましたら、九州から御参加の麻生先生、先生は意匠などいろいろ御研究されているかと思うのですけれども、先生が最近研究されている分野からみて、今日の議論はいかがでしょうか。

【麻生】　私自身はプログラムについても技術についても明るくないので、今回いろいろ勉強させていただいているというところです。

私の研究している分野から見てというところで余り思い浮かぶところはないのですが、今日のお話を伺っていますと、結局 10 条 3 項が確認規定だとして普通の著作物と同様に著作物性を判断すれば良いということであれば、そして、先ほどから御説明があるようにプログラムは書くときにかなり制限があるということであれば、それは創作性のところで不可避的な表現とか、そういうところに当たると思うのですけれども、私がよく分かっていないのは、Java API に限ったことで一般の API は違うのかもしれませんが、これは「ありふれた表現」とは何も関係してこないのかというところが気になるのですけれども、その辺りは議論にはならないのでしょうか。

【山神】　多分、説明の仕方として、こういう言語でプログラムを書いていって、そして例えば先ほどのソートをするプログラムの API といったことを考えたときに、平嶋先生もおっしゃっていたように、「java.lang.Math.max」といった名前を付けるというところから含めて、「普通はこうしますよね。」という書き方に落ち着いているわけですよね。

プログラムのコーディングをされる方は、奇抜なものを書くと後でメンテナンスをするとき、ほかの人に渡したときに大変なことになるので、皆さんやはり同調圧力というか、そろえた表現にしましょうということで、そういったことはいろいろな教科書でもいろいろな提案がされています。日曜プログラマーが日曜日に何か自分で適当に作って適当に自分で動かすというプログラムとは異なって、商品ベースのものだとかなり大規模になっていろいろな人が関わってくる。そうすると、やはり最大公約数的な表現で書かざるを得ないし、むしろそうあることが望ましいのだと。そうすると、恐らく、麻生先生がおっしゃったように、それはなるべくありふれたところで、違いが出てくるとしたら、多分解法が異なってくる。

ただ、解法も、異なった解法同士のコードだとやはり違うのですけれども、ある解法を採用したときの表現と別の人が同じ解法を採用したときに使うコードは、結局やはり似てくると。そうなると困るのでということで、多分 10 条 3 項などができたのかなという気がしています。

これでお答えになったかどうか分かりませんが、結論としては、「ありふれた表現」という説明というのはとても自然なことなのかなという気がしております。

【麻生】　ありがとうございました。

【奥邨】　すみません、今のところで少しよろしいでしょうか。

【山神】　どうぞ。

【奥邨】　私、先ほど「マージ」と言ってしまったのですけれども、正確に言うと、アメリカのこういう世界で昔よく言っていた「Scènes à faire」の方が近いのかもしれません。「ありふれた情景の理論」ということで、必ずしも一対一対応ではないとか、外部環境によって影響されるというのを少しマージとは分けて言ったときもあったので、日本法だと全部「ありふれた」ということになるのかもしれないのですけれども、アメリカ法で見るともう少し分けて議論するかもしれないと思いまして、その辺は先ほどざっくりまとめて説明してしまったので補足をしておきます。

【山神】　ありがとうございます。とても良い疑問だったということが奥邨先生からの御説明でも理解できました。大変ありがとうございます。

伊藤先生、お願いします。

【伊藤】　10 条 3 項の話をしているときに余り議論を拡散するのはどうかと思ってはいたのですが、別の切り口として、API の話でいきますと、2021 年の SOFTIC のセミナーでお話ししたときに、一つ一つの API 自体はありふれたものであって創作性が否定されてしまうのはやむを得ないかなと思うのですが、何千という API をパッケージ、クラス、メソッドといった体系に整理したということで、これ自体が編集著作物になって、その素材の選択配列を丸ごと Google がコピーしたことが編集著作物の著作権の侵害になるのではないかというのを一つ考えてはいたのですが、そういった議論が成り立ち得るのかどうかというのを一つ問題提起として挙げてみたいと思っておりました。

【山神】　ありがとうございます。多分 Oracle のウェブでも Java の API が公開されていて、その一覧を見る限りでは、場合によってはそれが編集著作物なのだと。記事の見出しとかいったものよりも整然と並んでいますけれども。

ただ、恐らく実際に Google がやったときには、それは API ごとにばらして使っていますので、そういう意味では、編集著作物だったとしても問題にならないということになるのではないでしょうか。

【伊藤】　さらには、実際のパッケージの整理の仕方をみると、開発者にとってなるべくあってほしいものや便利なものをなるべく網羅的にしらみつぶし的に作っているので、そういう意味でもその選択や配列に創作性が認めにくいのかと思ったのですが、今後例えば API 全般においてこういう議論がもしかしたらあり得るのかなと考えていた次第です。

【山神】　例えば、Google も Android のスマートフォンを開発するプログラマ向けにいろいろな技術資料を公開していまして、そこにも Android の中で使える仮想マシンの API、それは実質 Java にそろえてあるのだけれども、それを詳しく解説したドキュメントを公開したとします。多分どこかで公開はしていると思うのですけれども。

そうすると、それを捉えて、もし元のものが編集著作物ということになると侵害の問題となってくるかもしれませんけれども、伊藤先生の御理解のように、多分、網羅的に集めてしまうと、データベースの議論のときに出てきたように、限りなく著作物性が低くなっていくと。だから結局問題にならないのかなと。

ただ、問題になる局面というのは、そういう同じ技術資料を、対抗業者 A 社と B 社があって、A 社が技術的なデファクトスタンダードを確立して、それに乗っかる B 社が出てきて、そのときに開発者向けに同じような資料を公開して、そうするとパッと見はデッドコピーとは言いませんけれども、とても似ているということになる。それも結論としては侵害は否定する方向でやらないと困るので、そうすると、元の情報自体が編集著作物性が限りなく少ないという整理をするのが良いのかと思いました。

すみません、勝手にまとめてしまいましたけれども、伊藤先生、よろしいでしょうか。

【伊藤】　はい。

【山神】　ほかにいかがでしょうか。石新先生、どうぞよろしくお願いします。

【石新】　先生方の御議論を伺いながら一点だけ。

アメリカ法との比較なのですけれども、先ほど 10 条 3 項が創設規定か確認規定かという議論があったと思うのですが、今回の米国著作権法の 102 条(b)項の解釈のときに、Thomas 判事、そのほかの反対意見も、102 条(b)項自体を否定しているわけではなくて、102 条(b)項を前提としても宣言コードの表現の仕方としては Oracle が書いた書き方以外に書き方はあった、ほかに選択する表現があり得たということを理由に、102 条(b)項の存在を認めながら確認規定的な解釈をしたとして、原則マージ理論の前提としてもほかの表現の仕方があるから創作性はあるという理解をしていたと思います。

ということは、102 条(b)項がマージ理論だけではなくて、先ほどの椙山先生の議論に通じるところがあると思うのですけれども、技術的な内容にかなり近いコンピュータプログラムという特殊性からして、「method of operation」みたいなものであれば、method of operation を表現する仕方が幾らあろうが、それをどれほど創作的に表現していようが、method of operation である以上は著作物たり得ないと。それが 102 条(b)項なのだという、マージ理論にプラスアルファの 102 条(b)項の、創設規定的な解釈だと思うのですが、そういう主張も展開していました。

そうすると、10 条 3 項が確認規定だという日本法の理解からすると、仮に宣言コードが今回のケースよりももっと長い、非常に特殊な記載が多数あったということになると、確認規定的にいうと、ほかにも表現していたということもってして、マージせず、そこに創作性があるというべきだという、Thomas 判事みたいな解釈になり得ることにはならないだろうか・・・。

そうではなくて、10 条 3 項と 102 条(b)項を比較した・・・Thomas 判事みたいに宣言コードの書き方は幾つもあるのだから、それをコピーしたら、そこに創作性は見いだせるという話に・・・コピーをしたら著作権侵害になるのかどうなのかなと。私はそうではないのではないか、102 条(b)項の、創作的な、技術的な性格を重視して、method of operation のようなことをどれだけ創作的に表現してみても、そこに著作物性を認めるべきではないという立場に近い方が良いのではと思ったので、その点だけ少し御質問を。

【山神】　ありがとうございます。

つまるところは、宣言コードと言われている部分が数限りなく含まれていたのだと思うのですけれども、多数あったそれぞれをみても多分さほど長いものはなくて、石新先生がおっしゃるようにとても長い複雑なものになり得るものがあるのであればさておき、Java だけに限りませんけれども、多分一般のプログラムの世界でモジュールのインターフェース部分の定義の仕方というのはそこまで複雑化はしないのかという感触を持っています。そうすると、心配する必要はないのかなという気はします。

【石新】　素人の表現で良くなかったかもしれませんが、Google v. Oracle の事件が終わった直後の SOFTIC セミナーで伊藤先生と御一緒させていただいたときに、伊藤先生、うろ覚えで間違えていたら大変申し訳ないのですけれども、確か宣言コードの書き方なども意外といろいろ工夫の余地があって、書き方はほかにも幾つかあるというお話をされていて、だからこそ Thomas 判事な

どはそこに創作性を見いだしているわけですけれども、そうすると、確認規定的に読むと、Thomas 判事のように、今回のケースでも創作性ありという結論になるのではなかろうかと。複雑化して長くなったらどうなるのかという、少し変な方向に議論を引っ張ってしまいましたけれども、今回の Oracle のケースでさえ、Thomas 判事ほか、Oracle もそう主張している訳ですし・・・地裁で創作性ありと、ほかの表現を幾つか選択することが可能だったという一点を捉えて創作性があるというふうに言っているわけですけれども、10 条 3 項も確認規定的にほかの著作物と同じように、選択可能性で創作性を判断して良いかという議論はまた別にあるとして、ほかの表現を選択できる、2 つ 3 つであっても、幾つかあるということが言えれば、そこで創作性ありということになって、同じような結論にもなり得たのかなと。

複雑化という私の言い方が良くなかったのですが、創作性がより高いというイメージで申し上げたので、補足させていただきます。

【山神】　ありがとうございます。恐らく前々回くらいに議論になりましたが、始めに宣言コードの部分というか、それに相当する部分を設計するときには表現の選択の幅というのはあったと思うのですね。

ただ、一旦デファクトスタンダードとして確定してしまって、公開していくときに、それはどうなのかと。

おまけで言いますと、Java などの API などを整備していくときにはオープンソースコミュニティにいろいろ意見を聞きながら作り上げていくわけで、それで一旦それで決めてしまったと。ゲートボールの規則ではありませんけれども、ユニークなスポーツの規則とか、「えいや」と決めてしまった後にそれを使うという局面では、やはりもう選択の余地がなくなってしまうというのが今回のポイントだと思うのですね。

ですから、そこをアメリカでの判決の議論では少し見落としというか、勘違いなのかよく分かりませんけれども、そういうことなのかなというのが私の感触でありますけれども、奥邨先生いかがでしょうか。

【奥邨】　Thomas 判事が言っていることはウェラン対ジャスロー事件[10]的なのですよね。何も考えずに、「やろうと思ったら何でもできたのだからそれは表現だ。」と言っているということなのですが、正に先生がおっしゃったように、いろいろな業界の決まりごととか何とかを全く無視して言っているような気がしています。ですから、Thomas 判事は、コンピュータプログラムの議論で言えば、もう大昔に、最初に戻ったみたいなことを言っているような気がします。

今回で言えば、先ほど伊藤先生がおっしゃった、「API」と言ってしまうといろいろややこしいのですが、仕事の分割の仕方とか組合せにどんな名前を付けているかということは、もしかしたら編集著作物になるかもしれないのですが、そこは実は主張していないのですね。少なくとも最高裁はそこは議論になっていないと言っています。

そうだとすると、それはパブリックドメインとして今回は考えるということになると、既定のものを前提にどう組むかと考えると、前々回ぐらいに示していただいた Specification Book に、こういう名前を付けて、こういう順番でと書いてあって、名前ごと決まっているわけですから、そういう順番で決まっているときに Thomas 判事が言うように勝手に適当な名前を付けるとか、あのとおりにしないといけないものを、そこも無視して自由にやりようがあったのではないかというのは、もう随分昔の話かなという気がしながら読んでいて、そこは恐らく多数意見も入れなかったところなのではないかなというふうにも思いました。

ただ、細かく分類していくと、本当は元々の分類方法を反映しているところをどれくらい読み込むかという議論があるので、最高裁はそこから逃げてしまって、結局フェアユースで終わったのかなという気はしているので、全く議論がないとは思わないのですけれども、Thomas 判事はかなり大ざっぱな議論をしたような気がしておりまして、山神先生と同じようなイメージで伺っておりました。

【山神】　ありがとうございます。石新先生、いかがでしょうか。

【石新】　ありがとうございます。よく理解できました。ありがとうございます。

---

[10] *Whelan Associates, Inc. v. Jaslow Dental Laboratory, Inc.* 797 F.2d 1222 (3d Cir. 1986).

６　総括

（1） 本委員会の問題意識と第 2 回以降の研究対象について

初回となる第 1 回委員会（2022 年 10 月 31 日開催）では、委員会が「ハードウェア以外の全ての情報」を指す単語として「ソフトウェア等」を用いていることが委員会設置の趣旨説明の中で明らかにされ、幅広い情報財を対象とすることを前提に候補の検討が進められた。

最終的に、今年度は第 2 回から第 5 回を具体的な検討に、そして第 6 回を報告書案の検討に当てることとした。そこで扱うべきとされたのは、Google v. Oracle 事件[11]を素材とした Java API の宣言コード部分の創作性と仮に著作物性が肯定された場合の権利制限の在り方についてであった。

これは本委員会では、広く普及した Java やその互換開発環境にまつわる問題であり、世界規模でソフトウェア開発に与える影響が大きいことを理由とする。

幸いにも米国では権利行使が阻まれたが、仮に日本法で考えるとすればどのようになるかを明らかにすることは価値があると判断したのである[12]。

（2） 技術的背景としての Java API の位置づけと Java の標準化、OSS 化動向

第 2 回、3 回では、富士通の野山氏に Java の技術的内容を指導していただきながら、まずは、Java API とはどのようなものであったかについて理解を深め、その上で問題となった宣言コードの著作物性を議論した。もとより、Java API の位置づけと Java の標準化、OSS 化動向を理解することは制限規定を解釈する際にも有益であり、以下では筆者が別途収集した文献なども参照しつつ整理しておきたい。

まず注意すべきは、一般にプログラム開発で API(Application Programming Interface)といった場合には、あるライブラリ、モジュール（Java の場合はクラスライブラリ）の呼び出し方（呼出し規則、インターフェイス部分）が意識されるのであるが、これは Java 言語に限らず、また手続型かオブジェクト指向型かという言語パラダイムの違いにも左右されない。

しかしながら、本件で問題となっている「Java API」とは JRE (Java Runtime Environment)ひいては、JDK (Java Development Kit)に包含されるクラスライブラリ群を指す[13]。ともすれば、Google は Oracle が著作権を有する Java API を無断で流用したなどと書かれることもあるが、そもそも API という用語は Interface に重きを置く場合と、一般的なプログラムから繰り返し利用される再利用可能なモジュールを整備したものという 2 つの意味があり、それを区別しないといけないし、Google が流用したのは「Java API」と呼ばれる再利用可能なプログラムモジュール群の一部であったことに注意が必要である[14]。

当初の開発者である Sun は Java の開発に当たって、「write once, run anywhere.」なるスローガンを提唱しており[15]、これは現在でも一貫した方針として採用されている。 これは次のような仕組みで実現されている。

---

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

[12] 本判決を検討する日本語の論文としては、次のようなものがある。奥邨 弘司「Google v. Oracle 事件合衆国最高裁判決 : Java API を実現するプログラムのフェア・ユースについて」NBL1202 号 20 頁（2021）、大江修子＝彈塚 寛之「米国最高裁判例評釈 Google v. Oracle 事件最高裁判決 ：API 複製の著作権侵害性 ：Google LLC v. Oracle America、 Inc.、141 S.Ct. 1183(2021)」IP ジャーナル 19 号 69 頁（2021）、平嶋竜太「情報技術イノヴェーション促進と著作権エンフォースメントの調整法理としての fair use」パテント 75 巻 11 号（別冊パテント 27 号）171 頁（2022）、泉克幸 「プログラムにおける相互運用性の意義と著作権の保護範囲―Google 対 Oracle 事件米国最高裁判決を素材に」中央ロージャーナル 18 巻 4 号 3 頁（2022）。

[13] Java API 仕様は、以下から閲覧できる（2023 年 3 月 23 日最終閲覧）。
https://docs.oracle.com/javase/jp/8/docs/api/overview-summary.html

[14] Java SE を構成している API における 37 パッケージの宣言コード部分を Google が複製して、Android の開発プラットフォームに転用したものである。

[15] 886 F.3d at 1186.

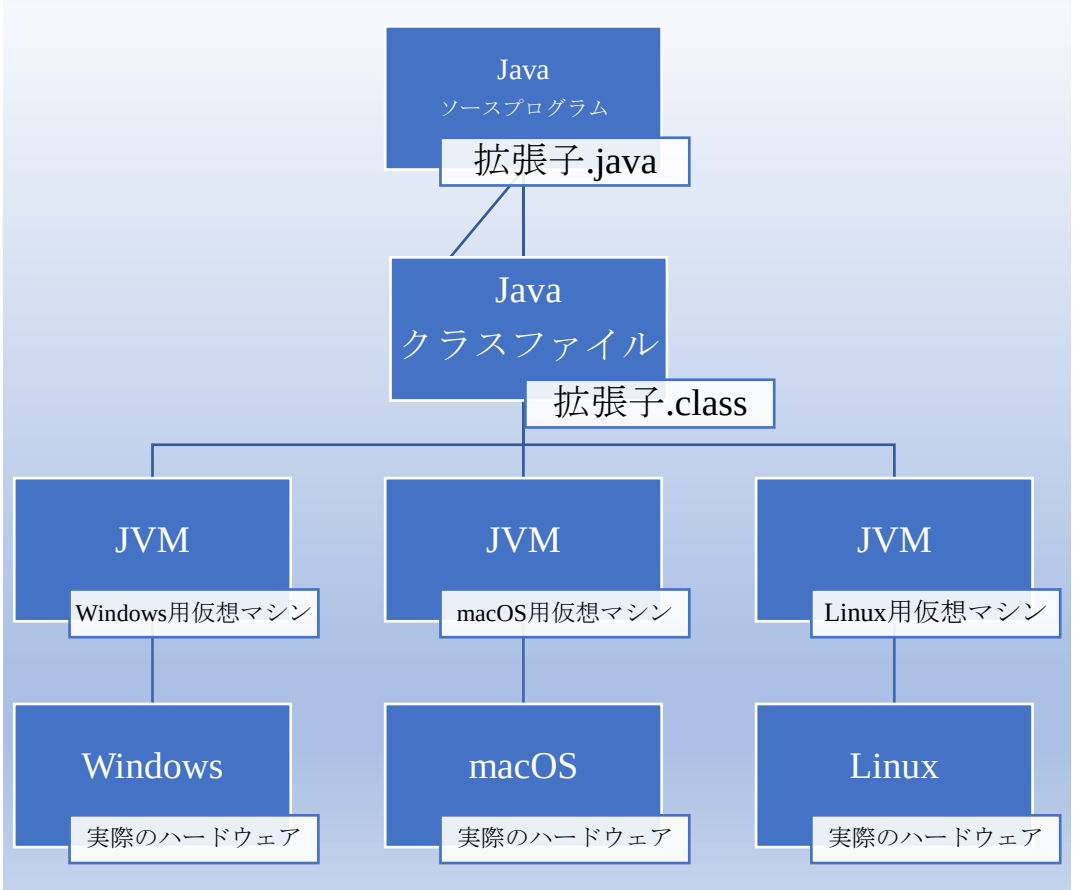

図　Java アプリケーションの動作の仕組み（筆者作成）

プログラムの開発者は、ソースプログラムを作りそれをコンパイルしてクラスファイルが出来上がる。クラスファイルの中身は、JVM 用のバイトコードであり、これは JVM 上で動作するので、各ハードウェア・OS 上で動作する JVM さえあれば環境を問わずプログラムが動作することになる。

一方 Google も Android のソフトウェア開発環境を構築する際に、Java の環境を使うことを考え、ライセンス交渉をしたものの決裂し、独自開発の道を突き進むこととなった。

Android 用アプリケーションとアプリケーション・フレームワークは、Google 独自に構築した仮想マシンである DalvikVM（後に Android Runtime（ART）へ発展）上の「Java Platform、 Standard Edition（Java SE）のサブセット+Android 独自拡張」環境で記述することを採用したのである。もっとも、"Android ネイティブ開発キット (Native Development Kit)" もリリースされているので、Java API を抜きにしてプログラムを開発することは不可能ではないが、Google が推奨しているのは Java によるアプリケーションのコーディングを基本とする方法となる。

本件訴訟提起時の Android 用アプリケーションは次のように動作していた（現在も大筋は同じである。）。

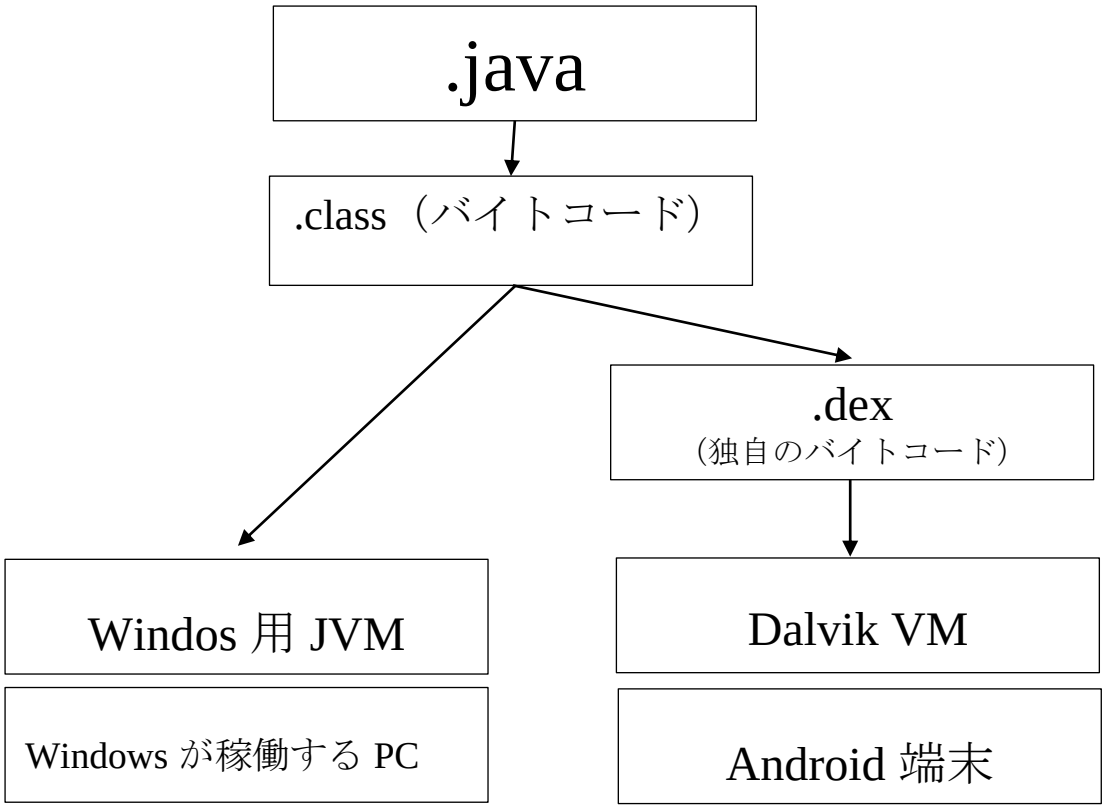

図　Android 用アプリケーションの仕組み　DEX

Java の開発環境である JDK は JRE (JVM+Java API) +コンパイラ・デバッガーであるが、Android の開発環境では、JVM を DalvikVM に、Java API を Java API サブセット+Android 独自の API へと置き換えているので、問題となったのは Java API サブセットの部分ということになる。

このような形で Google が独自路線に走った理由は、単にライセンス料の支払がいやであったわけではないようである。むしろ当初は Google もライセンスを受けようとしていたが、Sun(Oracle)が採用するライセンス形態の違いが大きかったようである。最高裁判決でも一定程度背景事情が認定事実として記されているが、技術者サイドから見た本件訴訟の背景事情をまとめた文献をもとに、以下整理したい。

Sun が生み出した Java が普及するにつれ、その標準化をオープンに策定するための団体が 1998 年に結成された。これが JCP (Java Community Process)である。
当初 JCP は企業しか参加できなかったが、JUG (Java Users Group)を介して個人も参加できるようになりその層が厚みを増した。

JCP の任務は Java の標準仕様を策定することで、実装は各社ごとに行うことになっていた。つまり、決められた標準に基づきベンダー各社が Java を実装する製品を販売できていた。こうすることにより、ベンダーが独自の仕組みを導入してユーザーを囲い込むことをしないようにすることができたのである。

そして、Java の標準仕様に従っているかどうかは Sun が提供する TCK (Technology Compatibility Kit)のチェックをパスすることで、Java の認定ロゴ（コーヒーマーク）を獲得することができた。一方 Java の仕様は、バージョンを追うごとにクラス数が増え、その結果 JavaVM のサイズなども肥大化していた。これは Java の部分的実装が認められなかったためである。

つまり、各社は各社ごとに実装できることには満足していたものの、実装の仕方の制約や Sun からの認定ロゴ取得コストが問題視されつつあったのである。

その頃に OSS の潮流に乗って登場したのが、1999 年設立の ASF (Apache Software Foundation)である。ASF が提供する Apache License 2.0 であり、このライセンスの元では「コードは商用製品やクローズド製品にも利用可能」で、「コードを書き換えたとしてもそれを公開しなくてもよい」と

いうメリットがあった。

これを最大限に利用したのが IBM であり、Java EE (Java Platform, Enterprise Edition)の世界では、例えば 2004 年には Tomcat が J2EE の Reference Implementation（RI：参照実装）となり、標準仕様だけではなくその標準的な実装を各社で共有することが盛んとなる。

しかしながら、Java SE の領域では、これがうまく行かなかったようである。具体的には、2005 年の Project Harmony の試みが物議を醸すことになるのである。

Project Harmony は J2SE[16] 5 の実装を Apache License 2.0 によって提供すること、そして、Java VM やクラスライブラリの実装を取捨選択、組み合わせて使うことができるようにすることを目指していた。

Sun はこれに難色を示し、J2SE を OSS 化することを提案した。これは後の OpenJDK につながる動きであったが、ライセンスが GPLv2 であり、改変したコードの公開義務があるため、ベンダー各社が独自の実装を追加することに障害があったのである。

2006 年 Apache Harmony が Java の認定を受けようとしたのであるが、TCK には「テスト対象となる JavaVM の実装が組み込み端末にも適用される場合、パスできない。」という利用制限により認証を拒絶されることになる。また、TCK は 2007 年に無償となるが、テスト対象は GPLv2 が適用されるものに限定された。

Apache は反発し、抗議活動を展開し、JCP でも Java の新しい仕様策定にことごとく反対するという動きに出た。

機を同じくして登場したのが、Android である。Google は 2005 年にスタートアップであった Android 社を買収し、携帯端末向けプラットフォームの開発に乗り出すことになる[17]。

当初 Google が構想していたのは、J2ME (Micro Edition)[18]の JavaVM の OSS 実装であったが、Apache Harmony をベースに開発していたので、ライセンス形態の違いから、認定を受けることが難しかった。そこでやむを得ず独自の JavaVM である DalvikVM を作り、そこに Apache Harmony のクラスライブラリから必要な部分をもってきて、独自のクラスライブラリと組み合わせた上で、2008 年には Apache License 2.0 で OSS 化することになる。

これに対して Sun は「Java の分断を招く。」と激しく非難したが、無料なのに自由に改変しそれを非公開にできるという仕組みはハードウェアメーカーなどのベンダーに支持され、現在の Android の隆盛がある。Java に関する Sun の収益モデルは、携帯端末など OEM ベンダーからのライセンス料に依存しており、異なるライセンス形態を取ることに加えて、クラスライブラリの実装を取捨選択、組み合わせて使うことができるようにすることは、Java の標準仕様をゆがめるもので、許し難いと Sun は考えた。以上が Java の熟練技術者から見たいわばオーラルヒストリー[19]を、判決文の事実認定も参照しつつ、筆者なりの言葉で整理した本事件の背景事情である。

以上、Google は単純にライセンス料の支払を避けるために DalvikVM を採用した訳ではなく、他にも幾つかの技術的理由があるものと思われる。

DalvikVM は、Java 仮想マシンに比べて、少ないメモリで動作する。当時の Android デバイスの RAM 容量が比較的少なかったため、この点は重要であった[20]。

また、そのような端末を対象とするが故に DalvikVM はなるべく高速で動作することが求められ

---

16 現在の Java SE に相当する。

17 886 F.3d 1179、 1187 (C.A. Fed. 2018); App. 137‑138、 242‑243.

18 携帯情報端末やデジタル家電、車載機器などの組み込み用途向けサブセット環境である。

19 日本 Java ユーザグループ (JJUG) は、Java 技術の向上・発展、開発者の支援を目的としたソフトウェアの開発者の任意団体である。そしてこの団体は定期的にセミナーを開催し、そのうち、2016 年 7 月に開催されたセミナーを 4 回に分けて報告しているが、そこでは、Java がどのように標準化されていき、Google がなぜそこからたもとを分かつことになったのかが、比較的詳細につづられている。（2023 年 3 月 24 日最終閲覧）
Google 対 Oracle の Java API 訴訟。歴史的経緯と IT 業界への影響を考える
https://www.publickey1.jp/blog/16/googleoraclejava_apiitjjug.html

20 Android が対象としていたスマートフォンは、Java ME が対象としていた携帯端末(今でいうガラケー)よりは、メモリ容量なども多かったが、パソコンなどと比べると圧倒的に貧弱なハードウェア環境であった。出村成和『クラウド活用のための ANDROID 業務アプリ開発入門』（日経 BP、2011）104 頁以下では、当時のスマートフォンの特徴、Android のメリットが列挙されており、とりわけ端末のメモリ、CPU の制約が指摘されている。

ていた[21]。そのため、Dalvik VM のアーキテクチャは、“レジスタ・ベース”を採用している。レジスタ・ベースとは、CPU 内部のレジスタにデータを格納して演算処理を実行する。一方、Java VM は“スタック・ベース”であるので、一時的なデータ格納領域（スタック）にデータを格納して演算処理を実行する。そして、“スタック・ベース”の VM は主メモリとのデータのやり取りが生じるため、CPU 内部のレジスタのみで実行する“レジスタ・ベース”の VM より速度が遅くなるのである。メモリの制約が大きく、CPU の性能も高くなかった当時、DalvikVM を開発することは理にかなったことであった。

(3) 宣言コード部分の性質と著作物性

第 2 回、第 3 回、そして第 6 回に著作物性についての議論を行ったが、そこでは次のような論点が議論の中心となった。

まず、本事件は地裁の判断が著作物性に否定的であるのに対して、高裁は肯定的であった。そして、最高裁があえて著作物性の判断を回避してフェアユースを採用したことは何を意味するであろうか。自信を持って著作物性を否定できるのであれば、わざわざフェアユースを持ち出すまでもないはずであり、判決で取り上げられた java.lang パッケージ内の Math クラスの max メソッドを呼び出す例であれば、奥邨委員の報告にあるように、宣言コード部分の著作物性はないという結論で異論はないであろう。

しかしながら、本委員会の議論でも、そもそも宣言コード部分はどのような意味を持つのか、宣言コードはそもそもプログラムなのか、そうではないのかについては様々な意見が出された。

結論として、宣言コード部分の各文言は Java 言語の文法によって規定されるところが大きく、自由度のある部分も最初の設計者が決めたものに制約されることが確認された[22]。換言すれば、Java SE とい Java プラットフォームを前提とするならば、Java API のインターフェイスと不即不離の表現であり、誤解を恐れずいうならば、Java 言語の特定プラットフォームにおける言語仕様とも言える。このような見方に立てば、我が国著作権法 10 条 3 項の言語や規約とも言えるようにも思われるし、筆者も当初はそのように考えていた[23]。

例えば、判決文で取り上げられた java.lang パッケージの Math クラスの各メソッドはいわゆるスタティックメソッドであり、クラスをインスタンス化しなくてもアクセスできるようになっている。逆に言えば、クラスをインスタンス化しなくてもアクセスできるようにしたいので、public static int max(int a, int b)という文字列としている[24]。max は最大値なので、自分が最初に決められる立場なら saidai でも構わないが、一旦 max と決められたなら、あとからそれと同等のものを作って差し替える場合に saidai としてしまうと、動かなくなってしまう。

また、クラスライブラリの中には様々なクラスファイルが含まれ[25]、そこには様々なメソッドなどが宣言されているが、奥邨委員の資料[26]にあるように宣言コードと実装コードが入り組んで存在することから、宣言コードはそもそもプログラムと言えるのかという点が議論になった。

分量的には、実装コードより宣言コードの方が圧倒的にサイズが小さいのではあるが、そのようなサイズの違いはさておき、両者は車の両輪に例えられるのであって、どちらかがかけたソース

---

[21] OS としての Android の仕組みついての日本語で書かれた文献として、次のものがある。　特に特に第 7 章「バイトコード実行環境 Dalvik と ART」では独自開発に向かった技術的背景が詳しく解説されている。有野和真『Android を支える技術〈I〉――60 fps を達成するモダンな GUI システム』（技術評論社、2017）。

[22] このような理解は、判決が前提とする事実関係とも一致する。Google が 37 のパッケージを流用せざるを得なかった事情は、141 S.Ct. at 1193f.を参照されたい。

[23] この点についてはこの後改めて検討する。

[24] ここで、static が static 修飾子で、これをつけることにより静的メソッドとなる。そして、public はアクセス修飾子と言われるもので、どこからアクセスできるかそのスコープを示す。int は返り値や引数の型が符号付き整数であることを示す型宣言のためのものである。

[25] 少々ややこしいが、ここで問題となっている Java API をコア API、標準 API、標準クラスライブラリと言ったりすることがあるが、全て同じものを指す。そして、クラスライブラリはクラスファイルをライブラリとしてまとめたものであるが、クラスファイルはソースコード(.java)をコンパイルして出来上がるファイル(.class)を指す。パッケージは、クラスライブラリとクラスファイルの中間に位置するもので、クラスファイルを機能の大きなくくりごとにひとまとめにしたものであり、オラクルのドキュメントでは、java.lang パッケージは「Java プログラミング言語の設計にあたり基本的なクラスを提供します。」と記載されている。（2023 年 3 月 24 日最終閲覧）
https://docs.oracle.com/javase/jp/8/docs/api/java/lang/package-summary.html

[26] 資料編 14 頁参照。

コードというものがあれば、そもそもコンパイルエラーでクラスファイルを作成することができない（そうするともちろんコア・ライブラリとして利用することもできない。）。

また、日本の著作権法においては、プログラムとは「電子計算機を機能させて一の結果を得ることができるようにこれに対する指令を組み合わせたものとして表現したもの」だとされるのであり（著 2 条 1 項 10 号の 2）、宣言コードの書き方によりコンパイラが生成するオブジェクトコードは変化するのであり、正に電子計算機を機能させる指令の組合せと言えよう。そうすると、宣言コードとはプログラムないしはプログラムの一部を取り出したものと理解できよう。

既に触れたとおり、最高裁は、著作物性の判断を回避したのであるが、仮に日本法で本件を考える際には Java API の(特定部分)の著作物性を考えるとするとどうなるであろうか。

日本の著作権法では、プログラム著作物を作成するために用いるプログラム言語、規約及び解法には著作権法による保護は及ばないものとされている（10 条 3 項）。

立法担当者の見解も学説も、10 条 3 項は、確認的規定であり、著作権法が表現を保護する法律である当然の帰結として、表現ではない思想・アイデアには著作権は及ばないことを定めた規定であるとしている点では一致する。

例えば立法担当者の見解は次の通りである[27]。

「本項は、プログラムの著作物に対する保護は、その作成のために用いられるプログラム言語、規約及び解法には及ばないことを念のため規定したものであります。著作物は、第 2 条第 1 項第 1 号の定義にありますように「表現したもの」でありますから、表現の手段として用いられる言語や、表現の背後にある原理、アイデア、約束事それ自体は著作物ではありません。また、著作物の保護の内容がこれらのものに及ぶものではありません。このことは当然のことではありますが、プログラムについては、著作権とりわけ翻案権の及ぶ範囲がその底流をなすアイデア等との関係でしばしば問題となるところから、その点を注意的に規定したものであります。」

コンピュータプログラムの表現された部分とは飽くまで、ソースコードとそれが変換されたオブジェクトコードであり[28]、そこに用いられるプログラム言語、規約、解法はいわばアイデアなのだから、著作権は及ばないという建て付けである。コンピュータプログラムの技術的性格からこれら表現とアイデアの境界線は曖昧となり、それを技術者にも分かる単語で確認的に示した点で意味のある規定と言えよう。

一方、コンピュータプログラムの保護に関する先駆的な研究でも、「インターフェースやプロトコールが文章的に表現されている場合、それを見てプログラムをコーディングしたとしても、それはアイディア（ルール）を利用しただけであり、規約に著作権が及ばない以上、著作権侵害にならないことは言うまでもない。問題となるのは、インターフェースやプロトコールが具体的なプログラムの形で記述されている場合である。互換機や端末機の製造あるいはデータ交換のためには、問題となるプログラムとインターフェースやプロトコールを合わせる必要があるため、当然に当該プログラムにアクセスすることになる。したがって、もしこのようなプログラムに著作物性が認められるとすれば、現実には著作物の類似性だけで侵害か否かが決まることになろう。

スポーツやゲームの場合であるならば、ルールそれ自体は著作物ではないが、それを具体的な表現としたルールブックが著作物である点に異論はないであろう。これとのアナロジーで考えるならば、プロトコールやインターフェースのアイディア（ルール）は著作物ではないが、それを具体化したプログラム自体は著作物ということになろう。しかしそうであるならば、あえて規約に著作権法の保護が及ばないと規定した意味がない。

そもそも、通常の著作物の場合と比し、インターフェースやプロトコールのプログラムは、あるルールを前提とする以上、その記述の自由度は極めて狭い。そもそもプログラム言語一般についても、自然言語に比べて選択の幅、すなわち自由度が著しく低いが、インターフェースやプロトコールのプログラムについては、それは現実問題として極度に低い。そのようなものに著作物性を認めると、著作権は形式的には表現の保護であっても実質的にはアイディアの保護となってしまうおそれがある。そこで、アイディアと表現とが密接な関係にあるプログラムの規約につき、それがいかなる形態であれ保護しないことを規定したのが、著作権法 10 条 3 項 2 号であると解すべきであろ

---

[27] 加戸守行『著作権法逐条講義〔7 訂新版〕』（著作権情報センター、2021）135 頁参照。なお、板東久美子「コンピュータ・プログラムに関する著作権法の一部改正について」 コピライト 292 号 7 頁も参照されたい。

[28] 本事件になぞらえるならば、.java と.class ないしは.dex となる。

う。」[29]と説かれる。やや長文の引用となったが、本委員会の検討においても、正にここに書かれているような議論が再三登場し、また同意を得られている。

もっとも、この記述については後に「中山信弘『ソフトウェアの法的保護〔新版〕』 46 頁では、インターフェースやプロトコールは、「それがいかなる形態であれ保護しないことを規定したのが著作権法 10 条 3 項 2 号であると解すべき」と記述したが、ミスリーディングな表現であった。インターフェースやプロトコールをプログラムとして表現してもすべからく著作権が発生しないということではなく、それらには他の選択肢がない、あるいは少ないために創作性が否定され、結果的に著作権が発生しない場合が多いということである」として補足が加えられている[30]。しかし、これらの記載は決して矛盾するものではない。『ソフトウェアの法的保護〔新版〕』の記述は、著作権法に翻案の概念が導入されたことにより、コンピュータプログラムなどの技術的性格を持つ著作物を保護することは、ともすればその背景にあるアイデアにまで保護を及ぼしてしまう危険性があり、それゆえ「言語・規約・解法には著作権の保護が及ばないと確認する利益は十分ある」と説明しようとしたものである[31]。

したがって、「インターフェースやプロトコールは、 それがいかなる形態で表現されていても、その背後にあるアイデアは保護しないことを規定したのが　著作権法 10 条 3 項 2 号であると解すべき」といように読み替えるならば、現時点においても何ら問題のない説明と言えよう。

なお、米国著作権法は「いかなる場合にも、著作者の創作的な著作物に対する著作権による保護は、アイデア、手順、プロセス、方式、操作方法、概念、原理又は発見には（これらが著作物において記述され、説明され、描写され、又は収録される形式を問わず）及ばない。」と定めるが[32]、この規定ぶりは日本法の 10 条 3 項の文言と整合的であり、日本法は、よりコンピュータプログラムの実態に即して特別に確認する規定を置いたものだと理解できる。

このほか、代表的な注解書でも同様の論旨が展開されているが[33]、これらの記載は手厚いものではなく、従来は余り検討されてこなかった領域であることが見て取れる。

そこで以下では、本件最高裁判決を受けて、改めて 10 条 3 項との関係やマージ理論の適用可能性、またありふれた表現に該当するかを検討した学説を紹介する。

例えば、「インターフェイスの保護と開放」と題して、業界標準として形成されたインターフェイスについて、競争法、消費者の便宜などの視点から多角的に検討すべきであると指摘しつつ、本件下級審から最高裁までを分析した文献[34]では、「我が国の著作権法では、10 条 3 項において、プログラムの著作物に対する保護は、プログラム言語や解法のほか、『規約』に及ばないことが明記されている。この『規約』は、通信を介し又は介さないで、あるプログラムを他の OS 又はアプリケーションプログラムと連動して、機能させるためにプログラムを表現する上で遵守すべきルールに従うことは、著作権侵害にはならない旨を確認的に規定したものであり、互換性を確保する上で必要となるインターフェイスや通信プロトコルの作成が不当に妨げられないよう配慮したものである。」、「他方、インターフェイスや通信プロトコルとして、現にプログラム表現されている一連のコード自体を複製することは、そこに著作物性が認められる限り、原則どおり著作権侵害を構成することが前提とされている。」「したがって、 『Oracle v. Google』事件のように、API の宣言コードの 1 万 1,500 行をそのまま複製した場合（verbatim）に、我が国の著作権法の適用をするとした場合、10 条 3 項により、直ちに適法であるとの結論が導かれるものではない。」とされている。

また、冒頭本件を分析した論文として紹介した本委員会委員でもある平嶋教授は、宣言コードが Java 言語というプログラム言語における「用法についての特別の約束」に近似（規約）に該当する

---

[29] 中山信弘『ソフトウェアの法的保護〔新版〕』（有斐閣、1988） 45 頁以下。なお、インターフェイス・プロトコルは規約との関係で分析されているが、この他、プログラム言語、解法に対も詳細に検討が加えられているのであるが、本事件の事実関係との関係から特にこの部分を抜き出して言及した。

[30] 中山信弘『著作権法〔第 3 版〕』（有斐閣、2020）138 頁注 144。

[31] 中山・前掲注(29) 49 頁注 1。

[32] 17 U.S.C. 102(b).

In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work.

[33] 例えば、小倉秀夫＝金井重彦編著『著作権法コンメンタール〔改訂版〕I』（第一法規、2020）362 頁〔森亮二〕、半田正夫＝松田 政行編著『著作権法コンメンタール〔第 2 版〕（1）』（勁草書房、2015）624 頁以下〔辛島　睦〕。

[34] 作花文雄『詳解著作権法〔第 6 版〕』（ぎょうせい、2022）863 頁以下。

のかを検討されている。

インターフェイスについては一般論として規約に該当するであろうとしつつ、「Java API のようにプログラミングに際して共通に用いられるソフトウエアの部品の集合体のようなものまで包含されるかは疑問であるとの立場から「Java API が言葉としてはインタフェースという呼称を含むものであることをもって直ちに著作権法上の「規約」と評価できるとは断じ得ない」とされる[35]。さらには、10 条 3 項に関する従来の学説と同様に、「技術的アイデア自体が著作物として保護されないとする考え方を踏まえた確認規定」であるとの考えを採用され、Java API 自体をもって技術的アイデア自体と評価することについては否定的であり、むしろ「Java API に包含される個々のプログラムのパーツを呼び出すために必要となる名称である宣言コードの部分についての複製行為であることに注意すべき」とされる。その上で、宣言コードの表現自体については用法についての特別の約束であるところの規約と考えることは難しいとされる[36]。

さらには、宣言コードが著作権法にいう「プログラム」に該当するのか、単なる「データファイル」なのかを分析され、IBF 事件[37]をひきつつ一見すると「データファイル」のようにも思われるが、結論として「プログラム」に該当し、著作物性がないものが多いが、その一部には著作物性があるのではないかとされる[38]。

一方、泉教授の分析によると、「こうした考え方を踏まえた上で、API の特性及び役割を考慮するならば、API が 10 条 3 項 2 号にいう『規約』に当たるとの解釈は合理的」、そうでないとしても「マージ理論の適用で創作性を否定」することが可能とされる[39]。

以上を踏まえて、Java API の各宣言コード部分はプログラム言語・規約・解法に相当するか改めて整理する。

野山氏の説明によれば、各宣言コード部分の記述は実質その通りに記述されない限り、互換システムの構築は不可能であり、一旦標準としてそれを定めたのであれば、いわば言語仕様とも規約（インターフェイス）とも言える（規約なのかプログラム言語なのかは、10 条 3 項が前記の通り確認規定である以上余り論ずる意味はないが、プログラム開発者の立場からすれば、自らが日常行っているプログラム開発作業において目にするプログラム言語、規約、解法のどれに当たるかは一応の判断を示しておいた方が、予測可能性が高まるように思われる。）。

筆者は当初、宣言コードはどちらかというと規約に近いのではないかと考えていたが、委員会での議論を通して、一応プログラム（ないしはその一部）であり、その表現は規約・プログラム言語と密接に関連するものであると理解すべきであるとの考えに至った。

規約やプログラム言語自体はアイデアであるが、それが具体的表現となれば著作物となり得るとの考え方は紹介した学説においても散見されるが、個別の宣言コードを、本件最高裁判決が取り上げていないものまで見ていったとしても、理論的には著作物性を有するものも存在する可能性を否定できないものの、実質的には、規約やプログラム言語と密接に結びついた表現（マージした表現ないしはありふれた表現）として、著作物性無しとしてしまってもよいように思われる。

そうすると既に紹介した作花教授の見解は、前半部分は従来のオーソドックスな学説と軌を一にするものであるが、末尾の本事件への当てはめについては、API の宣言コード 1 万 1,500 行がいわばひとまとまりであり、そこに著作物性がある可能性を念頭においているものと思われるが、そこが本委員会での分析とは少々異なる。

合計で 1 万 1,500 行と聞くと多いようにも思われるが、全体の 0.4%であったこと[40]、宣言コードは対応する実装コードと密接に結びつきつつ分散して存在すること、宣言コード自体は一応プログラム（の一部）ではあるが、極めて短いものが多いことを踏まえると結論としてその著作物性を否定してしまってよいのではないかというのが、本委員会での結論であった。

作花教授は、「法制度の公正性は時代の変化に即すべきであり、インターフェイス表現物が事実上の業界標準となり、その独占性の及ぶ範囲の認定に当たり全産業的な発展の視点が求められる余

35 平嶋・前掲注(12) 21 頁参照。
36 同上 22 頁。
37 東京高決平成 4 年 3 月 31 日知的裁集 24 巻 1 号 218 頁［IBF ファイル］。
38 平嶋・前掲注(12) 23 頁参照。
39 泉・前掲注(12)34 頁参照。
40 141 S.Ct. at 1205.

地は排除されるべきでない」と指摘されており[41]、平嶋教授も「一部の（著作物性がある）宣言コードについての複製権侵害を前提とした著作権行使、特に差止請求権の行使をそのまま肯定することは、Java API の一部の宣言コードの使用を排除することを意味するものであって、結果的に Android で動作するアプリケーション開発を Java 言語で行う行為自体を広く抑止・排除する効果にもつながる可能性も懸念され、妥当ではない」とされているのであり、仮に宣言コードの一部に著作物性を肯定しても、この後検討する制限規定の活用ないしは、競争法的な観点から規律が可能かもしれない。

(4) 制限規定の活用

本委員会は、本件最高裁判決が恐らく考慮した「宣言コードがわずかにでも著作物性を有し、その保護を認めることが好ましくない事態を招く」という状況を、日本的なフェアユースとしての制限規定を活用することで回避できないかという点についても検討を加えた。

本委員会での制限規定を活用した解釈については、奥邨委員、伊藤委員の報告を参照していただきたいが、簡単に整理すると次のようになるであろう。

まず、引用の規定を活用できるのではないかという斬新なアプローチが奥邨委員、伊藤委員のお立場であった。

仮に、日本で引用の規定を使うのであれば、近年スタンダードとなりつつある、条文の文言に素直な解釈としての「著作物を利用する側の利用の目的のほか、その方法や態様、利用される著作物の種類や性質、当該著作物の著作権者に及ぼす影響の有無・程度などが総合考慮されなければならない」をどのように本件の事実関係に当てはめていくかが問題となる。

そこでクローズアップされるのが、Dalvik の実装へと Google を向かわせた技術的背景である。この技術的背景は、奥邨委員の論文の言葉を借りるならば[42]、「問題は、法廷意見が言う、変容力のある利用目的、すなわち、スマートフォン用の新しいプラットフォームを作るという再実装目的が、32 条の定める「引用の目的」に含まれるか否か」という論点と大きく関わってくる。奥邨委員の言われる「再実装目的」に厚みを持たせるのが、「当時のスマートフォンの技術水準（少ないメモリ容量と低速の CPU）を有効に用いるためには、従来の肥大化しつつあった Java SE では不適切で、プラットフォームを再構築しつつ、従来のアプリやプログラム資産を活かすやり方しかなかった」という技術的背景なのであり、そのように考えると 32 条を活用するやり方は大いに可能性があると言えよう。

また、伊藤委員は 30 条の 4 を活用する道もあるのではないかと指摘されている（第 4 回 5 回の議論参照）。

平成 30 年改正で導入された 30 条の 4 は権利者が受ける不利益の度合いで 3 つに分けられる層のうちの第 1 層を対象として「著作物に表現された思想又は感情の享受を目的としない利用」を許すものである。そこで享受とは「著作物等の視聴等を通じて、視聴者等の知的・精神的欲求を満たすという効用を得ることに向けられた行為であるか否か」で判断されるとされているのであるが、プログラムの著作物については、「当該プログラムを実行等することを通じて、その機能に関する効用を得ることに向けられた行為であるかという観点から」判断されることになる。

本件で言えば、宣言コードの場合、当該部分を自己のプログラムに複製することで、コンピュータに解析・実行され、プログラムの機能に関する効用を得ることになるのであるから、享受利用とされるのではないであろうかというのが委員会としての結論であった。それゆえ 30 条の 4 は使えないということになる。

この点は、泉教授も 30 条の 4 第 3 項の適用について、奥邨委員の論文[43]を引用しつつ、適用は難しいとの立場を示されている[44]。

(5) まとめ（委員会としての結論）

Java、Kotlin 等によるソフトウェア開発環境の普及度合い、そしてそれらが 2006 年以降オープンソースコミュニティで構築されてきたという事実を踏まえるとすると、結論としては、制限規定と

---

41 作花・前掲注(34) 864 頁。
42 奥邨・前掲注(12)30 頁。
43 奥邨・前掲注(12)29 頁。
44 泉・前掲注(12)35 頁。

して引用（32 条）を活用する方法があり得、また、何らかの形で著作物性（とりわけ創作性）を否定する方法があり得る。その際には、10 条 3 項を直接活用する方法ではなく、その趣旨を踏まえた上で、当然の帰結として「表現がマージしている」ないしは「ありふれた表現である」ゆえに著作物性を否定することにより、日本法でも Google の行為を許容すべきであるということになりそうである。

著作権制度は、我が国においても権利者と利用者のバランスをとることが前提とされており、その上で文化の発展を目指すべきなのであるが[45]、仮に本件のような宣言コードの利用が制限されるとすると、一大産業となった Java において、既成の利益を優先する余り、創造的な表現の作出を促進するという基本的な要請がないがしろにされることになるのではないか。

したがって、最高裁判決においてはフェアユース理論の活用により、権利行使が抑止されたことは当然の帰結であり、いわゆる IT 業界の発展のためにも好ましいことであった。

最後に、10 条 3 項がなぜこのような文言となったのかについては、沿革がよくわからないということが委員会の検討では問題となった。時間の関係で、確かなことを記載することはできないが[46]、来年度以降委員会で検討することができればと考えている。

以上〔山神清和〕

[45] 「文化的所産の公正な利用に留意しつつ、著作者等の権利の保護を図り、もつて文化の発展に寄与する」（著作権法 1 条）。

[46] 文化庁著作権審議会第 2 小委員会（コンピューター関係）報告書（昭和 48 年 6 月）、文化庁著作権審議会第 6 小委員会（コンピュータ・ソフトウェア関係）中間報告(昭和 59 年 1 月)について精査してみても、その中には、「プログラム言語・規約・解法」に言及する部分はなく、昭和 60 年の改正で突然入ったように見受けられる。もっとも、「コンピュータプログラムに係る著作権問題に関する　調査研究協力者会議報告書　―既存プログラムの調査・解析等について―」（平成 6 年 5 月）には興味深い記述が見受けられた。すなわち、同報告書の「はじめに」では、我が国におけるプログラムの著作権保護は、「言語、規約及び解法が保護の対象から除外されている」ゆえに、欧米に比べて水準が低いとの批判があり、それに対する反論として、10 条 3 項の趣旨は「著作権は表現を保護しアイデアを保護するものではないという基本原則をプログラムについて確認的に規定しているものと解されている。この基本原則は、国際的にも承認されている」と述べているのである。そして、国際的にも承認されているとの証左として、EC ディレクティブ第 1 条第 2 項や既に紹介した米国著作権法第 102 条（b）が同様の規定であることを挙げている。であるとするならば、10 条 3 項の「プログラム言語・規約・解法」という文言は、欧米のこれらの規定を参考にしつつ、プログラム開発に携わる技術者にヒヤリングした上で、コンピュータプログラムにふさわしい文言として選ばれたのではなかろうか。

第３　今後の検討について

今年度の調査研究委員会では、米国における Google v. Oracle 事件における議論を素材として、コンピュータプログラムに関する著作物性や権利制限の在り方、すなわち権利保護の在り方等に関して、我が国著作権法も含め、詳細な検討を行った。

2023 年 3 月 23 日に開催された第 6 回委員会では、来年度において本調査研究委員会で取り上げる論点・議題について検討したところ、今年度の調査研究委員会における議論も踏まえつつ、引き続き「著作物性」、「創作性」の観点を中心に置いて議論を進めることとされた。

2023 年に入ってからは、画像の生成を行う「Stable Diffusion」や文章の生成を行う「ChatGPT」に代表される「生成系 AI（Generative AI）」と呼ばれるシステムが急速に実用化されるとともに我が国を含む世界各国で急速に注目を集め、それに伴い、それらが生成する画像、文章等と著作権法との関係が現実的かつ喫緊の課題として浮上してきている。

生成系 AI が抱える著作権法上の課題として少なくとも以下の四つの点が問題となるであろう。

１　生成系 AI が生成した情報が著作物となるか

著作権法では、「著作物」とは「思想又は感情を創作的に表現したものであつて、文芸、学術、美術又は音楽の範囲に属するもの」であると定義されている（著作権法 2 条 1 項 1 号）。ここで「思想又は感情」とは、「（動物などではなく）人間の思想又は感情」であるとされているので[47]、従来は、生成系 AI が生成した情報は、人間の思想や感情を反映してはおらず、結論として著作物ではないとされてきた。しかしながら、出来上がった文章やイラストなどは、典型的な著作物と見分けが付かないものが多く、これを著作物として保護しないという結論は受け入れ難いように思われる。そこで、AI の操作者に着目して創作性を検討する考え方も示唆されていたが[48]、それでよいのかどうかは検討する必要がある。また、AI が生成する情報が常に創作性を有する訳ではないので、創作性の概念についての再検討も必要であろう。

２　仮に生成系 AI が生成した情報が著作物である場合、その著作権は誰に帰属するか

この場合、当該情報の表現のうち、創作性のある部分を生成することに寄与した者は誰なのかということを考える必要があり、最終的な表現生成までに、AI モデルの開発者、学習データの提供者、AI の利用者（操作者）が関与しているのであるから、その候補となる。特定の行為者を創作者と判定するためには、創作行為とは何か、突き詰めて言えば、創作的表現を作り出すとはどのような行為であるのかを見つめ直す必要がある。前述の通り、AI の操作をした者が著作者になり、その者に著作権が帰属するという見解が有力ではあるが、そのような理解でよいのか、検討が必要である。

３　生成系 AI が他者の著作物を参照や模倣して情報を生成する場合、その行為は著作権侵害に当たるのかどうか

生成系 AI は、既存の情報から学習することで、当該情報の表現上の特徴を抽象化して新しい情報を生成することが多い。そういった行為はアイデアの流用でしかなく、著作権法では許容されている。ただその際に、他者の著作物から直接的又は間接的に引用や借用を行うならば、侵害を惹起する可能性が高くなる。しかし、現行の著作権法では、引用等についても一定の場合にそれを許容している、引用が許されるかどうかの境界線には曖昧な部分が多い。また、著作権法が保護しないアイデアと保護する表現との境界も曖昧であり、何が創作的表現であるかの正しい理解がない限り、結論を出せない。

４　AI による既存の著作物の利用と著作権法の制限規定

AI が質の高い情報を生み出すには、その前提として AI モデルが高度に学習されている必要があり、そのために既存の情報（多くは著作物である）を学習用データとして利用する。我が国においては、このような利用行為は、著作権法 30 条の 4 の規定により、「著作物に表現された思想又は感情の享受を目的としない利用」であるとして、著作権者の許諾なく行えることになっているが、

[47] 中山・前掲注(30) 50 頁参照。
[48] 中山・前掲注(30) 79 頁参照。

権利者側の反発は強い。AI の各モデルに即した情報生成のメカニズムに立ち返って、このような制限規定を維持すべき説得的な理由を示せない限り、特定の著作物を学習から外したり、学習データとして利用する著作物に使用料を支払うことを求められたりしかねない（既にそのような主張がなされている）。このようなことを認めてしまうと、せっかく設けられた先進的な著作権法の制限規定が骨抜きにされ、ひいては文化の発展、産業の発達を阻害することになるであろう。

以上のように、生成系 AI が抱える著作権法上の課題は多岐にわたり、かつ、創作性とは何かという点についての検討なしには、結論が出せないと言えよう。

本調査研究委員会では、法と技術との関係や著作権法を始め各知的財産権法制に造詣の深い専門家を委員として多数擁する特徴を踏まえ、来年度、「著作物性」、「創作性」を掘り下げて検討するという課題に取り組むこととしたい。

〔山神清和、SOFTIC 調査研究部〕

ソフトウェア等の権利保護に関する調査研究報告書 － 2022（令和 4）年度 －

2023（令和 5）年 3 月発行

発行所　一般財団法人ソフトウェア情報センター

〒105-0003 東京都港区西新橋 3-16-11

Tel: 03-3437-3071

E-mail: res@softic.or.jp