**Tugas 1**

Pemrograman Berorientasi Objek

Gilang Permana | 20200040041 | TI20 Regular A

### 

### **Buatlah resume(ulasan) informasi tentang versi terakhir JAVA saat ini! Jelaskan fitur-fitur yang ditambahkan beserta kelebihan-kelebihannya.**

Java 17, rilis dukungan jangka panjang (LTS) baru dari Java standar, sekarang tersedia untuk penggunaan produksi. Oracle juga mengumumkan bahwa rilis LTS, yang setidaknya akan menerima delapan tahun dukungan produk, selanjutnya akan tiba setiap dua tahun, berbeda dengan tiga tahun antara rilis di masa lalu. Rilis non-LTS mendapatkan dukungan enam bulan dari Oracle.

Di antara kemampuan baru dalam versi baru Java standar adalah dukungan filter deserialisasi khusus konteks, yang merupakan peningkatan keamanan, dan pratinjau pencocokan pola untuk pernyataan sakelar. JDK 17 menampilkan semua yang telah ditambahkan sejak rilis LTS terakhir, JDK 11, yang tiba tiga tahun lalu.

Rilis LTS yang lebih sering akan memberikan akses lebih cepat ke fitur baru untuk perusahaan yang hanya ingin menggunakan rilis LTS, kata Georges Saab, wakil presiden grup platform Java Oracle. Rilis LTS berikutnya adalah Java 21 pada tahun 2023. Dengan JDK 17, Oracle akan mengizinkan penggunaan binari Oracle JDK secara gratis dalam produksi selama tiga tahun, satu tahun setelah LTS berikutnya. Tapi itu tidak termasuk langganan dukungan produksi perusahaan.

Data dari basis pelanggan penyedia pemantauan aplikasi New Relic, yang mewakili puluhan juta JVM produksi, menunjukkan bahwa rilis LTS memiliki penyebaran yang hampir bulat. New Relic menemukan bahwa hampir 100 persen pengguna menjalankan JDK 11 atau JDK 8, dua rilis LTS terbaru. New Relic mengatakan 90 persen menjalankan JDK 11 dan 10 persen JDK 8.

Namun, Oracle mengatakan unduhan dari rilis enam bulan terus meningkat. Pengembang suka mencoba rilis enam bulan sementara perusahaan ingin menyebarkan rilis LTS.

Pembuatan produksi JDK 17 dapat ditemukan di oracle.com. Build open source OpenJDK juga tersedia. Fitur-fitur baru JDK 17 antara lain sebagai berikut:

* [**Context-specific deserialization filters**](https://openjdk.java.net/jeps/415)memungkinkan aplikasi untuk mengonfigurasi filter deserialisasi khusus konteks dan dipilih secara dinamis melalui pabrik filter seluruh JVM yang dipanggil untuk memilih filter untuk setiap operasi serialisasi. Dalam menjelaskan motivasi di balik proposal ini, Oracle mengatakan bahwa deserializing data yang tidak tepercaya adalah aktivitas yang berbahaya secara inheren karena konten aliran data yang masuk menentukan objek yang dibuat, nilai bidangnya, dan referensi di antara mereka. Dalam banyak kegunaan, byte dalam aliran diterima dari klien yang tidak dikenal, tidak dipercaya, atau tidak diautentikasi. Dengan konstruksi aliran yang hati-hati, musuh dapat menyebabkan kode di kelas arbitrer dieksekusi dengan maksud jahat. Jika konstruksi objek memiliki efek samping yang mengubah status atau meminta tindakan lain, tindakan tersebut dapat membahayakan integritas objek aplikasi, objek pustaka, dan runtime Java. Kunci untuk menonaktifkan serangan serialisasi adalah untuk mencegah contoh kelas arbitrer dari deserialized, sehingga mencegah eksekusi langsung atau tidak langsung dari metode mereka. Filter deserialization diperkenalkan di Java 9 untuk mengaktifkan aplikasi dan kode library untuk memvalidasi aliran data yang masuk sebelum deserializing mereka. Kode ini menyediakan logika validasi sebagai **java.io.ObjectInputFilter** saat membuat aliran deserialisasi. Namun, mengandalkan pembuat streaming untuk meminta validasi secara eksplisit memiliki keterbatasan. JDK Enhancement Proposal 290 mengatasi keterbatasan ini dengan memperkenalkan JVM-wide deserialization yang dapat diatur melalui API, properti sistem, atau properti keamanan, tetapi pendekatan ini juga memiliki batasan, terutama dalam aplikasi yang kompleks. Pendekatan yang lebih baik adalah mengonfigurasi filter per streaming sehingga tidak memerlukan partisipasi setiap pembuat streaming. Peningkatan yang direncanakan akan membantu pengembang membangun dan menerapkan filter yang sesuai untuk setiap konteks deserialisasi dan kasus penggunaan.
* Dengan **restoration of always-strict floating point semantics**, operasi floating-point akan dibuat ketat secara konsisten, daripada memiliki semantik floating point yang ketat (strictfp) dan semantik floating point default yang agak berbeda. Ini mengembalikan semantik floating point asli ke bahasa dan VM, mencocokkan semantik sebelum pengenalan mode floating point yang ketat dan default di Java Standard Edition 1.2. Tujuan dari upaya ini termasuk memudahkan pengembangan perpustakaan yang sensitif secara numerik termasuk **java.lang.Math** dan **java.lang.StrictMath.** Dorongan untuk mengubah semantik floating point default di akhir 1990-an berasal dari interaksi buruk antara bahasa Java asli dan semantik JVM dan beberapa keanehan set instruksi koprosesor floating point x87 dari arsitektur x86 yang populer. Mencocokkan semantik floating point yang tepat dalam semua kasus, termasuk operan dan hasil subnormal, memerlukan overhead instruksi tambahan yang besar. Mencocokkan hasil tanpa adanya overflow atau underflow dapat dilakukan dengan lebih sedikit overhead dan kira-kira itulah yang diizinkan oleh semantik floating point default yang direvisi yang diperkenalkan di Java SE 1.2. Tetapi ekstensi SSE2 (Streaming SIMD Extensions 2), yang dikirimkan dalam prosesor Pentium 4 dan yang lebih baru mulai sekitar tahun 2001, dapat mendukung operasi floating point JVM yang ketat secara langsung tanpa overhead yang tidak semestinya. Sejak Intel dan AMD mendukung SSE2 dan ekstensi yang lebih baru yang memungkinkan dukungan alami dari semantik floating point yang ketat, motivasi teknis untuk memiliki semantik floating point default yang berbeda dari yang ketat tidak ada lagi.
* **Deprecation of the Security Manager,** mempersiapkan penghapusan di rilis mendatang. Kembali ke Java 1.0, Manajer Keamanan telah menjadi sarana utama untuk mengamankan kode Java sisi klien dan jarang digunakan untuk mengamankan kode sisi server. Tujuan proposal adalah mengevaluasi apakah API atau mekanisme baru diperlukan untuk menangani kasus penggunaan sempit tertentu yang telah menggunakan Manajer Keamanan, seperti memblokir **System::exit**. Rencana menyerukan penghentian Pengelola Keamanan untuk dihapus bersamaan dengan Applet API lama, yang juga dijadwalkan tidak digunakan lagi di JDK 17.
* Pratinjau **pattern matching for switch** memperluas bahasa pola di Java untuk memungkinkan ekspresi dan pernyataan **switch** diuji terhadap sejumlah pola, masing-masing dengan tindakan tertentu. Hal ini memungkinkan kueri berorientasi data yang kompleks untuk diekspresikan secara ringkas dan aman. Di antara tujuan fitur ini termasuk memperluas ekspresi dan penerapan ekspresi dan pernyataan **switch** dengan mengaktifkan pola untuk muncul dalam label kasus, melonggarkan historis null-hostility switch saat diinginkan, dan memperkenalkan dua jenis pola: **guarded patterns,** yang memungkinkan pola logika pencocokan untuk disempurnakan dengan ekspresi Boolean arbitrer, dan **parenthesized patterns**, yang menyelesaikan beberapa ambiguitas penguraian. Di JDK 16, operator instanceof diperluas untuk mengambil pola tipe dan melakukan pencocokan pola. Ekstensi sederhana yang diusulkan memungkinkan idiom instanceof-and-cast yang sudah dikenal disederhanakan.
* **Strong encapsulation for JDK internals,** kecuali untuk API internal penting seperti sun.misc.Unsafe, tidak akan memungkinkan lagi untuk mengendurkan enkapsulasi elemen internal yang kuat melalui opsi baris perintah tunggal, seperti yang dapat dilakukan di JDK 9 hingga JDK 16. Sasaran rencana termasuk meningkatkan keamanan dan pemeliharaan JDK dan mendorong pengembang untuk bermigrasi dari elemen internal ke API standar.
* **Removal of the Remote Method Invocation (RMI) Activation mechanism,** Sambil melestarikan sisa RMI. Mekanisme Aktivasi RMI sudah usang dan tidak digunakan dan tidak digunakan lagi untuk dihapus di JDK 15.
* **foreign function and memory API,** memperkenalkan tahap inkubator, memungkinkan program Java untuk beroperasi dengan kode dan data di luar runtime Java. Dengan menjalankan fungsi asing secara efisien, yaitu kode di luar JVM, dan mengakses memori asing dengan aman, yaitu memori yang tidak dikelola oleh JVM, API memungkinkan program Java untuk memanggil pustaka asli dan memproses data asli tanpa kerapuhan dan risiko JNI (Java Antarmuka Asli). API yang diusulkan adalah evolusi dari dua API — API akses memori asing dan API tautan asing. Foreign memory access API ditargetkan ke Java 14 pada 2019 sebagai API inkubasi dan diinkubasi kembali di Java 15 dan Java 16. Foreign linker API ditargetkan ke Java 16 sebagai API inkubasi pada akhir 2020. Tujuan dari rencana API termasuk kemudahan penggunaan, kinerja, umum, dan keamanan.
* Diintegrasikan ke dalam JDK 16 sebagai inkubating API, platform-agnostik Vector API akan diinkubasi lagi di JDK 17, menyediakan mekanisme untuk mengekspresikan komputasi vektor yang dikompilasi secara andal pada waktu proses ke instruksi vektor optimal pada arsitektur CPU yang didukung. Ini mencapai kinerja yang lebih baik daripada perhitungan skalar yang setara. Di JDK 17, API vektor telah ditingkatkan untuk kinerja dan implementasi, termasuk peningkatan untuk menerjemahkan vektor byte ke dan dari array boolean.
* **Sealed classes and interfaces,** membatasi kelas atau antarmuka mana yang dapat diperluas atau diimplementasikan. Tujuan proposal termasuk memungkinkan pembuat kelas atau antarmuka untuk mengontrol kode mana yang bertanggung jawab untuk mengimplementasikannya, menyediakan cara yang lebih deklaratif daripada pengubah akses untuk membatasi penggunaan superclass, dan mendukung arah masa depan dalam pencocokan pola dengan menyediakan landasan untuk analisis pola yang lengkap. Kemampuan ini membantu desainer API membuat kode yang lebih tangguh.
* **Removal of the experimental AOT and JIT compiler,** yang jarang digunakan tetapi membutuhkan upaya pemeliharaan yang signifikan. Rencana tersebut memerlukan pemeliharaan antarmuka kompiler JVM tingkat Java sehingga pengembang dapat tetap menggunakan versi kompiler yang dibuat secara eksternal untuk kompilasi JIT. Kompilasi AOT (alat jaotc) dimasukkan ke dalam JDK 9 sebagai fitur eksperimental. Alat ini menggunakan **kompiler Graal,** yang ditulis dalam Java, untuk kompilasi AOT. Fitur eksperimental ini tidak disertakan dalam build JDK 16 yang diterbitkan oleh Oracle dan tidak ada yang mengeluh. Di bawah rencana yang ditentukan, tiga modul JDK akan dihapus: jdk.aot (alat jaotc); internal.vm.compiler, kompiler Graal; dan jdk.internal.vm.compiler.management, Graal MBean. Kode HotSpot yang terkait dengan kompilasi AOT juga akan dihapus.
* **Porting JDK ke MacOS/AArch64** sebagai tanggapan atas rencana Apple untuk mentransisikan komputer Macintosh dari x64 ke AArch64. Port AArch64 untuk Java sudah ada untuk Linux dan pekerjaan sedang berlangsung untuk Windows. Pembangun Java berharap untuk menggunakan kembali kode AArch64 yang ada dari port ini dengan menggunakan kompilasi bersyarat, seperti norma di port JDK, untuk mengakomodasi perbedaan dalam konvensi tingkat rendah seperti antarmuka biner aplikasi dan set register prosesor yang dicadangkan. Perubahan untuk MacOS/AArch64 berisiko merusak port Linux/AArch64, Windows/AArch64, dan MacOS/x64 yang ada, tetapi risikonya akan dikurangi melalui pengujian praintegrasi.
* **Menghentikan Applet API untuk dihapus.** API ini pada dasarnya tidak relevan, karena semua vendor browser web telah menghapus dukungan untuk plugin browser Java atau telah mengumumkan rencana untuk melakukannya. Applet API sebelumnya tidak digunakan lagi, tetapi tidak untuk dihapus, di Java 9 pada September 2017.
* **Pipeline rendering baru untuk MacOS,** menggunakan Apple Metal API sebagai alternatif dari pipeline yang ada yang menggunakan OpenGL API yang tidak digunakan lagi. Proposal ini dimaksudkan untuk menyediakan pipeline rendering yang berfungsi penuh untuk Java 2D API yang menggunakan kerangka MacOS Metal dan siap jika Apple menghapus OpenGL API dari versi MacOS mendatang. Pipeline ini dimaksudkan untuk memiliki paritas fungsional dengan pipeline OpenGL yang ada, dengan kinerja yang baik atau lebih baik dalam aplikasi dan benchmark tertentu. Arsitektur bersih akan dibuat yang sesuai dengan model Java 2D saat ini. Pipa akan hidup berdampingan dengan pipa OpenGL sampai usang. Bukan tujuan proposal untuk menambahkan API Java atau JDK baru.
* **Generator nomor pseudo-acak** yang disempurnakan yang akan menyediakan tipe antarmuka baru dan implementasi untuk generator nomor pseudorandom (PRNG) termasuk PRNG yang dapat dilompati dan kelas tambahan algoritma PRNG yang dapat dipisah (LXM). Antarmuka baru, **RandomGenerator,** akan menyediakan API seragam untuk semua PRNG yang ada dan yang baru. Empat antarmuka RandomGenerator khusus akan disediakan. Memotivasi rencana tersebut adalah fokus pada beberapa bidang untuk perbaikan di bidang pembuatan nomor pseudorandom di Jawa. Upaya ini tidak memerlukan penyediaan implementasi dari banyak algoritma PRNG lainnya. Tetapi tiga algoritma umum telah ditambahkan yang sudah banyak digunakan di lingkungan bahasa pemrograman lainnya. Tujuan dari rencana tersebut meliputi :
  + Mempermudah penggunaan berbagai algoritma PRNG secara bergantian dalam aplikasi.
  + Peningkatan dukungan untuk pemrograman berbasis aliran, menyediakan aliran objek PRNG.
  + Penghapusan duplikasi kode di kelas PRNG yang ada.
  + Pelestarian perilaku yang ada dari kelas java.util.Random.