Erdianti Wiga Putri Andini

13522053 – K1

TUGAS BACA

SECTION 1.4, 1.6, 1.7, 1.8 HOLGER GAST

1.4. Methods

Pemrograman berorientasi objek (OOP) sangat bergantung pada metode dalam kelas untuk memfasilitasi kolaborasi dan definisi perilaku. Metode-metode ini erat terkait dengan kelas, dengan instansi diciptakan dan metode dipanggil pada instansi-instan tersebut. Polimorfisme, termasuk penimpaan metode dan overloading, meningkatkan fleksibilitas dengan memungkinkan objek mengambil bentuk yang berbeda dan memungkinkan kelas turunan menggantikan metode yang diwarisi dengan versi baru. Fitur-fitur ini berkontribusi pada ekspresivitas dan adaptabilitas kode.

Metode-metode memainkan berbagai peran, termasuk sebagai penyedia layanan, antarmuka kenyamanan, langkah-langkah pemrosesan, alat penjelasan, dan penanganan panggilan kembali, meningkatkan keterbacaan dan fungsionalitas kode. Alat-alat refactoring Eclipse membantu dalam ekstraksi metode dan restrukturisasi, menyederhanakan proses perbaikan kualitas kode secara iteratif. Penggunaan kembali kode adalah fundamental untuk efisiensi, dengan metode memainkan peran krusial sebagai dasar penggunaan kembali, menawarkan solusi sebagai layanan kepada klien.

Pewarisan memperluas kemungkinan penggunaan kembali dengan memungkinkan kelas turunan membangun fungsionalitas yang sudah ada, tetapi mengidentifikasi layanan yang dapat digunakan kembali dengan efektif memerlukan perencanaan yang hati-hati. Prinsip-prinsip pengembangan Agile menganjurkan untuk memulai dengan solusi yang sederhana dan melakukan refactoring sesuai kebutuhan, menekankan peningkatan bertahap dari waktu ke waktu. Sebagai alternatif, mengekstraksi fungsionalitas yang dapat digunakan kembali ke objek terpisah mempromosikan fleksibilitas dan enkapsulasi, meningkatkan pemeliharaan dan pemahaman kode.

Memahami konsep identitas, kesetaraan, dan hashing sangat penting untuk mengelola objek secara efektif dalam OOP. Pengujian untuk identitas objek menggunakan `==` mencerminkan semantik referensi, sementara objek nilai membandingkan kesetaraan berdasarkan isi mereka. Mengimplementasikan metode `equals()` dan `hashCode()` penting untuk perbandingan yang konsisten dan pengambilan yang efisien dalam struktur data berbasis hash.

Meskipun ada alat otomatis untuk menghasilkan metode `equals()` dan `hashCode()`, pengembang harus memahami arti pentingnya untuk perilaku yang benar, terutama dalam skenario seperti pengumpulan sampah yang efisien atau penanganan objek yang berpindah. Situasi wasiat yang ditolak, di mana sebuah kelas turunan tidak dapat menyediakan layanan yang diwarisi dengan efisien, mungkin memerlukan penanganan yang hati-hati, seperti melempar pengecualian. Namun, ada konteks yang dapat diterima untuk wasiat yang ditolak, seperti menciptakan objek palsu yang ringan untuk tujuan pengujian. Secara keseluruhan, pemahaman menyeluruh tentang identitas, kesetaraan, dan hashing

sangat penting untuk pengelolaan objek yang efektif dan menjaga perilaku yang konsisten dalam sistem perangkat lunak.

1.6. Constructors

Konstruktor merupakan bagian fundamental dalam pemrograman berorientasi objek, bertanggung jawab atas inisialisasi objek saat pembuatan dengan mengubah memori mentah menjadi entitas yang koheren dan siap digunakan. Mereka menetapkan keadaan awal dan menjaga invarian objek, memastikan konsistensi. Konstruktor default umum, memungkinkan pembuatan objek tanpa argumen, sedangkan konstruktor berparameter menawarkan fleksibilitas. Namun, konstruktor sebaiknya dieksekusi dengan cepat, menghindari operasi yang panjang. Teknik inisialisasi yang malas menunda pengaturan kompleks hingga diperlukan, mengoptimalkan kinerja awal, terutama dalam aplikasi berskala besar.

Langkah inisialisasi eksplisit kadang diperlukan sepanjang siklus hidup objek, terutama dalam sistem reflektif atau yang dapat diperluas. Metode siklus hidup seperti `init()` memberikan kontrol eksplisit atas inisialisasi, memastikan pengaturan yang tepat pada waktu yang sesuai. Metode-metode ini, sering kali didefinisikan dalam kelas dasar abstrak, mempromosikan inisialisasi yang konsisten di seluruh subkelas. Dalam hirarki pewarisan, konstruktor memastikan inisialisasi bidang yang tepat di setiap tingkat. Meskipun konstruktor tidak diwariskan, mereka dapat menyerahkan tanggung jawab inisialisasi kepada konstruktor kelas induk melalui panggilan `super()`, dengan memperhatikan untuk menghindari mengakses bidang yang belum diinisialisasi dalam metode-metode yang di-override.

Konstruktor juga memfasilitasi penyalinan objek dan menawarkan metode konstruktor statis untuk meningkatkan keterbacaan dan fleksibilitas dalam pembuatan objek, terutama di mana overloading mungkin menyebabkan kebingungan. Peran mereka dalam inisialisasi objek sangat penting untuk mempertahankan integritas objek dan mendukung berbagai strategi inisialisasi. Melalui desain dan penggunaan yang cermat, konstruktor memastikan objek diinisialisasi dengan benar dan siap digunakan di berbagai skenario pemrograman.

1.7. Packages

Di Java, packages memainkan peran penting dalam mengelola namespace dan menentukan cakupan visibilitas. Mereka mencegah tabrakan nama dengan menyediakan struktur hierarkis untuk mengorganisir kelas-kelas dalam perpustakaan, kerangka kerja, atau proyek-proyek. Organisasi ini memudahkan pengembangan dan pemeliharaan yang efektif dengan memisahkan kode ke dalam namespace yang berbeda-beda. Namun, perlu diperhatikan untuk menghindari penggunaan kembali nama kelas di packages yang berbeda, karena dapat menyebabkan kebingungan dan mengurangi keterbacaan kode untuk klien.

Selain itu, packages berfungsi sebagai komponen, mencerminkan keputusan desain pada tingkat objek dan kelas. Mereka memungkinkan penciptaan unit fungsional yang koheren,

dengan mengemas objek-objek terkait dalam satu namespace yang umum. Dengan menawarkan lingkungan terkendali di mana objek-objek berkolaborasi tanpa gangguan eksternal, packages meningkatkan modularitas dan kegunaan kembali. Visibilitas default dalam packages membatasi akses ke komponen internal, mempromosikan enkapsulasi, dan melindungi dari penggunaan yang tidak disengaja.

Selain itu, Pola Fasad adalah prinsip desain berharga yang digunakan untuk menyederhanakan subsistem yang kompleks dengan menyediakan antarmuka yang terpadu. Fasad menawarkan API yang jelas dan tingkat tinggi yang mengabstraksikan detail kerja internal yang rumit, sehingga membuat subsistem lebih mudah digunakan untuk klien. Dalam konteks pengembangan Java, kerangka kerja seperti Eclipse memanfaatkan Pola Fasad secara luas untuk menawarkan akses intuitif ke berbagai fungsionalitas seperti manipulasi kode Java dan pengembangan plugin, dengan demikian meningkatkan kesederhanaan desain perangkat lunak dan pemeliharaannya.

1.8. Basics of Using Classes and Objects

Dalam pemrograman berorientasi objek, objek merupakan komponen vital yang mengkapsulasi data dan perilaku, berfungsi sebagai blok bangunan dari sistem perangkat lunak. Mereka mewakili entitas dunia nyata atau konsep abstrak, menekankan tugas-tugas tertentu dan berkolaborasi dengan yang lain untuk mencapai tujuan yang lebih luas. Prinsip desain berorientasi objek menekankan pembuatan objek yang kecil, terfokus, dan logis yang menjaga informasi, menjalankan tindakan, dan membuat keputusan.

Objek memenuhi berbagai peran seperti pemegang informasi, penyedia layanan, pengendali, pengatur, pemberi antarmuka, dan koordinator dalam sebuah sistem. Pemegang informasi mengelola data tanpa menerapkan logika, sementara penyedia layanan menjalankan algoritma tertentu dan menyediakan fungsionalitas untuk objek lain. Pengendali membuat keputusan dan mengatur operasi yang kompleks, sementara pengatur mengelola hubungan objek. Pemberi antarmuka menghubungkan komponen sistem ke entitas eksternal, dan koordinator mengawasi interaksi antara objek.

Penyedia layanan memainkan peran penting dengan menawarkan layanan yang dibutuhkan oleh objek lain, mempromosikan kelas yang dapat digunakan kembali dan memfasilitasi penggunaan yang efektif melalui perencanaan yang hati-hati. Objek batas bertindak sebagai penjaga, memvalidasi data dan permintaan masuk sebelum melewatkan mereka ke sistem inti, sehingga menjaga kebersihan dan kesederhanaan sistem. Mereka juga memfasilitasi terjemahan format data eksternal ke format internal, memastikan adaptabilitas tanpa memengaruhi fungsionalitas inti. Kelas bersarang, termasuk kelas anggota dalam dan statis, membantu dalam menstrukturkan objek yang bekerja sama secara erat dalam sebuah sistem, meningkatkan keterbacaan dan keberlanjutan. Namun, kelas anonim harus digunakan dengan bijaksana, dan penggunaan objek nol harus didekati dengan hati-hati karena potensi kompleksitas. Pendekatan alternatif seperti melempar pengecualian atau menggunakan pola Objek Nol harus dipertimbangkan untuk mengurangi masalah yang terkait dengan nilai nol.