LAPORAN ANALISIS LCOM DAN CBO DALAM SISTEM MARKETPLACE

Oleh:

Ihsan Fauzi 241524048

PROGRAM STUDI D4-TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

BAB I
PENDAHULUAN

Dalam pengembangan software terutama di bahasa java, dua metrik LCOM (Lack Cohesion of Methods) dan CBO (Coupling Between Objects) sangat penting untuk evaluasi OOP.

1. **LCOM (Lack of Cohesion of Methods)**
   Mengukur sejauh mana metode-metode dalam suatu kelas yang tidak saling berhubungan, yang menunjukkan bahwa kelas tersebut mungkin memiliki terlalu banyak tanggung jawab dan melanggar Single Responsibility Principle.

2. **CBO (Coupling Between Objects)**
   Mengukur tingkat ketergantungan antar kelas, di mana nilai CBO yang tinggi menunjukkan bahwa kelas-kelas tersebut sangat tergantung satu sama lain, sehingga sulit untuk dimodifikasi secara independen.

Dokumen ini bertujuan menganalisis beberapa kasus LCOM dan CBO dalam sistem marketplace dan memberikan rekomendasi untuk meningkatkan kualitas kode.

BAB II
ANALISIS LCOM (LACK OF COHESION IN METHODS)

A. Kasus Admin terlalu banyak peran
   1. Kode yang bermasalah :

```
public class Admin {
    private List<Customer> customers;
    private List<Seller> sellers;

    public void moderateReviews(Review review) {
        System.out.println("Moderasi review: " + review.getContent());
    }

    public void submitReview(Review review) {
        System.out.println("Admin mengirim review (seharusnya bukan tugas Admin)");
    }

    public void processPayment(double amount) {
        System.out.println("Memproses pembayaran sebesar: " + amount);
    }

    public void applyDiscount(Customer customer, double discount) {
        System.out.println("Menerapkan diskon " + discount + "% untuk " +
customer.getName());
    }

    public void monitorSecurity() {
        System.out.println("Memantau keamanan sistem...");
    }

    public void detectFraud() {
        System.out.println("Mendeteksi aktivitas mencurigakan...");
    }

    public void trackUserActivity() {
        System.out.println("Melacak aktivitas pengguna...");
    }

    public void sendNotification(String message) {
        System.out.println("Mengirim notifikasi: " + message);
    }
}
```

   2. Mengapa masuk LCOM
      Kelas Admin memiliki peran yang terlalu banyak dan bercampur:
      a. Moderasi (moderateReviews, submitReview)
      b. Pembayaran (processPayment, applyDiscount)

c. Keamanan (monitorSecurity, detectFraud)
d. Analisis (trackUserActivity)
e. Notifikasi (sendNotification)

Metode-metode ini tidak selalu menggunakan atribut yang sama:

a. processPayment() tidak berkaitan dengan moderateReviews()
b. trackUserActivity() tidak berhubungan dengan applyDiscount()

3. Solusi

```
public class FinanceAdmin {
    public void processPayment(double amount) {
        System.out.println("Memproses pembayaran sebesar: " + amount);
    }

    public void applyDiscount(Customer customer, double discount) {
        System.out.println("Menerapkan diskon " + discount + "% untuk " +
customer.getName());
    }
}

public class ContentModerator {
    public void moderateReviews(Review review) {
        System.out.println("Moderasi review: " + review.getContent());
    }
}

public class SecurityAdmin {
    public void monitorSecurity() {
        System.out.println("Memantau keamanan sistem...");
    }

    public void detectFraud() {
        System.out.println("Mendeteksi aktivitas mencurigakan...");
    }
}

public class NotificationService {
    public void sendNotification(String message) {
        System.out.println("Mengirim notifikasi: " + message);
    }
}
```

B. Kasus Seller dengan metode di luar tanggung jawab

1. Kode yang bermasalah :

```
public class Seller {
    private String name;

    public void sendNotification(String message) {
        System.out.println("Seller mengirim notifikasi: " + message);
    }

    public void manageOrder(Order order) {
        System.out.println("Seller mengelola pesanan: " + order.getId());
    }
}
```

2. Mengapa masuk LCOM
    - Notifikasi dan Manajemen Order sebenarnya bukan merupakan tanggung jawab utama Seller
    - Seharusnya Admin atau sistem khusus yang mengelola order dan mengirim notifikasi
    - Seller seharusnya hanya berfokus pada produk dan transaksi
3. Solusi

```
public class NotificationService {
    public void sendNotification(String message) {
        System.out.println("Mengirim notifikasi: " + message);
    }
}

public class OrderManager {
    public void manageOrder(Order order) {
        System.out.println("Mengelola pesanan: " + order.getId());
    }
}

public class Seller {
    private String name;
    // Sekarang Seller hanya fokus pada tugasnya
}
```

C. Kasus Customer dengan metode keamanan

1. Kode yang bermasalah :

```
public class Customer {
    private String name;
    private double balance;

    public void detectFraud() {
        System.out.println("Customer mendeteksi penipuan (seharusnya bukan tugas
Customer)");
    }
}
```

2. Mengapa masuk LCOM
    - Keamanan dan deteksi fraud seharusnya merupakan tanggung jawab SecurityAdmin
    - Customer seharusnya hanya berinteraksi dengan transaksi dan profilnya sendiri
3. Solusi

```
public class SecurityAdmin {
    public void detectFraud() {
        System.out.println("Admin mendeteksi aktivitas mencurigakan...");
    }
}

public class Customer {
    private String name;
    private double balance;
    // Sekarang Customer tidak lagi memiliki tanggung jawab yang bukan miliknya
}
```

BAB III
ANALISIS COUPLING BETWEEN OBJECT (CBO)

A. Kasus Admin dengan Akses Langsung ke Saldo Pengguna

1. Kode yang bermasalah

```
public class Admin {
    public void checkBalance(Customer customer, Seller seller) {
        System.out.println("Saldo Customer: " + customer.getBalance());
        System.out.println("Saldo Seller: " + seller.getBalance());
    }
}
```

2. Mengapa masuk CBO
   - Admin mendapatkan informasi yang bukan haknya, yaitu saldo Customer dan Seller
   - Admin terlalu bergantung pada struktur internal Customer dan Seller, menyebabkan coupling tinggi
3. Solusi

```
public class BalanceService {
    public void verifyBalance(String userId, double requiredAmount) {
        // Verifikasi saldo tanpa memberikan nilai eksak
        System.out.println("Memverifikasi kecukupan saldo user: " + userId);
    }
}

public class Admin {
    private BalanceService balanceService;

    public Admin(BalanceService balanceService) {
        this.balanceService = balanceService;
    }

    public void verifyFunds(String customerId, double amount) {
        balanceService.verifyBalance(customerId, amount);
    }
}
```

B. Kasus Seller dapat melihat transaksi terakhir Customer

1. Kode yang bermasalah

```
public class Seller { public void viewLastTransaction(Customer customer) {
System.out.println("Transaksi terakhir Customer: " + customer.getLastTransaction()); } }
```

2. Mengapa masuk CBO
    - Seller mendapatkan informasi yang bukan haknya, yaitu transaksi terakhir dari Customer
    - Ini meningkatkan coupling karena Seller sekarang tergantung langsung pada struktur Customer
3. Solusi

```
public class TransactionManager {
    public boolean verifyTransactionExistence(String customerId, String transactionId) {
        // Verifikasi transaksi tanpa melihat detailnya
        System.out.println("Memverifikasi keberadaan transaksi: " + transactionId);
        return true;
    }
}

public class Seller {
    private TransactionManager transactionManager;

    public Seller(TransactionManager transactionManager) {
        this.transactionManager = transactionManager;
    }

    public void checkTransactionStatus(String customerId, String transactionId) {
        boolean exists = transactionManager.verifyTransactionExistence(customerId,
transactionId);
        System.out.println("Transaksi " + (exists ? "ada" : "tidak ada"));
    }
}
```

C. Kasus Admin dapat melihat daftar produk yang belum dirilis oleh Seller

1. Kode yang bermasalah

```
public class Admin {
    public void viewUnreleasedProducts(Seller seller) {
        System.out.println("Produk yang belum dirilis: " + seller.getUnreleasedProducts());
    }
}
```

2. Mengapa masuk CBO
    - Admin tidak seharusnya mengetahui produk yang belum dirilis oleh Seller, kecuali ada peraturan khusus atau persetujuan sebelumnya.
    - Ini membuat Admin bergantung pada detail internal Seller.

3. Solusi
   Gunakan sistem persetujuan (approval) yang memungkinkan Seller mengajukan produknya tanpa langsung terekspos ke Admin.

BAB IV
KESIMPULAN

Berdasarkan analisis LCOM dan CBO yang telah dilakukan, dapat disimpulkan bahwa:

1. Untuk Masalah LCOM:
   - Pemisahan tanggung jawab sangat penting untuk meningkatkan kohesi kelas
   - Setiap kelas harus fokus pada satu tanggung jawab spesifik sesuai dengan Single Responsibility Principle
   - Pemisahan Admin menjadi beberapa kelas seperti ContentModerator, FinanceAdmin, dan SecurityAdmin meningkatkan kohesi dan maintainability

2. Untuk Masalah CBO:

   - Mengurangi ketergantungan langsung antar kelas dengan menggunakan service layer atau interface
   - Menerapkan prinsip encapsulation untuk menyembunyikan informasi yang tidak perlu diekspos
   - Menggunakan pattern seperti Mediator atau Facade untuk mengelola interaksi antar objek

Dengan menerapkan solusi yang diusulkan, kualitas kode akan meningkat secara signifikan, menjadikan sistem lebih mudah dikelola, diuji, dan dikembangkan di masa mendatang.

BAB V
SOLUSI KESELURUHAN SOURCE CODE

```
import java.util.*;

// Core model: User as abstract base class
abstract class User {
    protected String username;
    protected String email;

    public User(String username, String email) {
        this.username = username;
        this.email = email;
    }

    public String getUsername() {
        return username;
    }

    public String getEmail() {
        return email;
    }

    public void login() {
        System.out.println(username + " telah login.");
    }

    public void logout() {
        System.out.println(username + " telah logout.");
    }

    abstract void performAction();
}

// Financial concerns moved to separate service
class FinancialService {
    private Map<String, Double> userBalances = new HashMap<>();

    public void addUser(String username, double initialBalance) {
        userBalances.put(username, initialBalance);
    }
```

```
    public boolean hasEnoughBalance(String username, double amount) {
        Double balance = userBalances.getOrDefault(username, 0.0);
        return balance >= amount;
    }

    public void adjustBalance(String username, double amount) {
        Double currentBalance = userBalances.getOrDefault(username, 0.0);
        userBalances.put(username, currentBalance + amount);
        System.out.println("Saldo " + username + " disesuaikan sebesar $" + amount);
    }

    public void processPayment(String fromUser, String toUser, double amount) {
        if (hasEnoughBalance(fromUser, amount)) {
            adjustBalance(fromUser, -amount);
            adjustBalance(toUser, amount);
            System.out.println("Pembayaran $" + amount + " dari " + fromUser + " ke " + toUser);
        } else {
            System.out.println("Saldo tidak mencukupi untuk " + fromUser);
        }
    }
}

// Notification concerns moved to specialized service
class NotificationManager implements NotificationService {
    @Override
    public void sendNotification(String recipient, String message) {
        System.out.println("Notifikasi untuk " + recipient + ": " + message);
    }
}

interface NotificationService {
    void sendNotification(String recipient, String message);
}

// Product management moved to dedicated service
class ProductService {
    private Map<String, Map<String, Double>> sellerProducts = new HashMap<>();
    private Map<String, List<String>> unreleasedProducts = new HashMap<>();

    public void registerSeller(String sellerUsername) {
```

```
        sellerProducts.putIfAbsent(sellerUsername, new HashMap<>());
        unreleasedProducts.putIfAbsent(sellerUsername, new ArrayList<>());
    }

    public void addProduct(String sellerUsername, String product, double price) {
        sellerProducts.get(sellerUsername).put(product, price);
        System.out.println(sellerUsername + " menambahkan produk: " + product + " seharga $" + price);
    }

    public void addUnreleasedProduct(String sellerUsername, String product) {
        unreleasedProducts.get(sellerUsername).add(product);
        System.out.println(sellerUsername + " menambahkan produk draft: " + product);
    }

    public boolean isProductAvailable(String sellerUsername, String product) {
        return sellerProducts.getOrDefault(sellerUsername, new HashMap<>()).containsKey(product);
    }

    public double getProductPrice(String sellerUsername, String product) {
        return sellerProducts.getOrDefault(sellerUsername, new HashMap<>()).getOrDefault(product, 0.0);
    }

    public List<String> getApprovedUnreleasedProducts(String sellerUsername, String adminUsername) {
        // Only return if proper approval in place
        System.out.println(adminUsername + " mendapatkan produk draft yang disetujui dari " + sellerUsername);
        return unreleasedProducts.getOrDefault(sellerUsername, new ArrayList<>());
    }
}

// Order management moved to dedicated service
class OrderService {
    private Map<String, List<Order>> userOrders = new HashMap<>();
    private int nextOrderId = 1000;

    public OrderService() {
        for (ShippingStatus status : ShippingStatus.values()) {
            // Initialize all statuses
        }
    }

    public void registerUser(String username) {
```

```
        userOrders.putIfAbsent(username, new ArrayList<>());
    }

    public String createOrder(String buyerUsername, String sellerUsername, String product, double price) {
        String orderId = "ORD" + nextOrderId++;
        Order order = new Order(orderId, buyerUsername, sellerUsername, product, price);
        userOrders.get(buyerUsername).add(order);
        System.out.println("Pesanan dibuat: " + orderId + " untuk " + product);
        return orderId;
    }

    public void updateOrderStatus(String orderId, ShippingStatus status) {
        // Find and update order
        System.out.println("Status pesanan " + orderId + " diperbarui ke " + status);
    }

    public List<Order> getSellerOrders(String sellerUsername) {
        List<Order> sellerOrders = new ArrayList<>();
        for (List<Order> orders : userOrders.values()) {
            for (Order order : orders) {
                if (order.getSellerUsername().equals(sellerUsername)) {
                    sellerOrders.add(order);
                }
            }
        }
        return sellerOrders;
    }

    public List<Order> getBuyerOrders(String buyerUsername) {
        return userOrders.getOrDefault(buyerUsername, new ArrayList<>());
    }
}

class Order {
    private String id;
    private String buyerUsername;
    private String sellerUsername;
    private String product;
    private double price;
    private ShippingStatus status;
```

```
    public Order(String id, String buyerUsername, String sellerUsername, String product, double price) {
        this.id = id;
        this.buyerUsername = buyerUsername;
        this.sellerUsername = sellerUsername;
        this.product = product;
        this.price = price;
        this.status = ShippingStatus.PENDING;
    }

    public String getId() {
        return id;
    }

    public String getBuyerUsername() {
        return buyerUsername;
    }

    public String getSellerUsername() {
        return sellerUsername;
    }

    public String getProduct() {
        return product;
    }

    public double getPrice() {
        return price;
    }

    public ShippingStatus getStatus() {
        return status;
    }

    public void setStatus(ShippingStatus status) {
        this.status = status;
    }
}

enum ShippingStatus {
    PENDING, PROCESSING, SHIPPED, DELIVERED
}
```

```
// Review system moved to dedicated service
class ReviewService {
    private List<Review> reviews = new ArrayList<>();
    private List<Review> pendingReviews = new ArrayList<>();

    public void submitReview(String username, String product, String content, int rating) {
        Review review = new Review(username, product, content, rating);
        pendingReviews.add(review);
        System.out.println(username + " mengirimkan ulasan untuk " + product + ": " + content);
    }

    public void moderateReview(String adminUsername, Review review, boolean approved) {
        if (approved) {
            pendingReviews.remove(review);
            reviews.add(review);
            System.out.println(adminUsername + " menyetujui ulasan: " + review.getContent());
        } else {
            pendingReviews.remove(review);
            System.out.println(adminUsername + " menolak ulasan: " + review.getContent());
        }
    }

    public List<Review> getPendingReviews() {
        return new ArrayList<>(pendingReviews);
    }

    public List<Review> getApprovedReviews() {
        return new ArrayList<>(reviews);
    }
}

class Review {
    private String username;
    private String product;
    private String content;
    private int rating;

    public Review(String username, String product, String content, int rating) {
        this.username = username;
        this.product = product;
```

```
        this.content = content;
        this.rating = rating;
    }

    public String getUsername() {
        return username;
    }

    public String getProduct() {
        return product;
    }

    public String getContent() {
        return content;
    }

    public int getRating() {
        return rating;
    }
}

// Security concerns moved to dedicated service
class SecurityService {
    private Set<String> monitoredUsers = new HashSet<>();

    public void monitorUser(String username) {
        monitoredUsers.add(username);
        System.out.println("Memantau keamanan akun: " + username);
    }

    public boolean detectFraud(String username, double amount) {
        boolean suspicious = amount > 1000;
        if (suspicious) {
            System.out.println("Aktivitas mencurigakan terdeteksi untuk " + username + ": $" + amount);
        }
        return suspicious;
    }
}

// Discount handling moved to dedicated service
class DiscountService {
```

```
    private Map<String, Double> userDiscounts = new HashMap<>();

    public void setUserDiscount(String username, double discountPercentage) {
        userDiscounts.put(username, discountPercentage);
    }

    public double applyDiscount(String username, double price) {
        double discountPercentage = userDiscounts.getOrDefault(username, 0.0);
        double discountAmount = price * (discountPercentage / 100);
        System.out.println("Diskon sebesar $" + discountAmount + " diterapkan untuk " + username);
        return price - discountAmount;
    }
}

// Shipping concerns moved to dedicated service
class ShippingService {
    public void shipOrder(String sellerUsername, String orderId) {
        System.out.println(sellerUsername + " sedang mengirim pesanan: " + orderId);
    }

    public void trackShipment(String orderId) {
        System.out.println("Melacak pengiriman untuk pesanan: " + orderId);
    }
}

// Loyalty program moved to dedicated service
class LoyaltyService {
    private Map<String, Integer> userPoints = new HashMap<>();

    public void addUser(String username) {
        userPoints.put(username, 0);
    }

    public void addPoints(String username, int points) {
        int currentPoints = userPoints.getOrDefault(username, 0);
        userPoints.put(username, currentPoints + points);
        System.out.println(username + " mendapatkan " + points + " poin loyalitas");
    }

    public boolean redeemPoints(String username, int points) {
        int currentPoints = userPoints.getOrDefault(username, 0);
```

```
        if (currentPoints >= points) {
            userPoints.put(username, currentPoints - points);
            System.out.println(username + " menukarkan " + points + " poin loyalitas");
            return true;
        }
        return false;
    }

    public int getPoints(String username) {
        return userPoints.getOrDefault(username, 0);
    }
}

// Analytics service for tracking user activities
class AnalyticsService {
    public void trackUserActivity(String username, String activity) {
        System.out.println("Aktivitas tercatat: " + username + " - " + activity);
    }

    public void generateReport(String adminUsername) {
        System.out.println(adminUsername + " membuat laporan analitik");
    }
}

// Customer class now focused only on customer-specific operations
class Customer extends User {
    private List<String> purchaseHistory = new ArrayList<>();

    public Customer(String username, String email) {
        super(username, email);
    }

    @Override
    void performAction() {
        System.out.println(username + " sedang berbelanja di e-commerce.");
    }
}

// Seller class now focused only on seller-specific operations
class Seller extends User {
    public Seller(String username, String email) {
```

```
        super(username, email);
    }

    @Override
    void performAction() {
        System.out.println(username + " mengelola toko online.");
    }
}

// Admin class now focused only on admin-specific operations
class Admin extends User {
    public Admin(String username, String email) {
        super(username, email);
    }

    @Override
    void performAction() {
        System.out.println(username + " mengelola sistem e-commerce.");
    }
}

// Main system class that coordinates the services
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class ECommerceSystem {
    private FinancialService financialService;
    private NotificationManager notificationManager;
    private ProductService productService;
    private OrderService orderService;
    private ReviewService reviewService;
    private SecurityService securityService;
    private DiscountService discountService;
    private ShippingService shippingService;
    private LoyaltyService loyaltyService;
    private AnalyticsService analyticsService;

    private Map<String, User> users = new HashMap<>();

    public ECommerceSystem() {
```

```
        this.financialService = new FinancialService();
        this.notificationManager = new NotificationManager();
        this.productService = new ProductService();
        this.orderService = new OrderService();
        this.reviewService = new ReviewService();
        this.securityService = new SecurityService();
        this.discountService = new DiscountService();
        this.shippingService = new ShippingService();
        this.loyaltyService = new LoyaltyService();
        this.analyticsService = new AnalyticsService();
    }

    public void registerCustomer(String username, String email) {
        Customer customer = new Customer(username, email);
        users.put(username, customer);
        financialService.addUser(username, 0);
        orderService.registerUser(username);
        loyaltyService.addUser(username);
        System.out.println("Pelanggan baru terdaftar: " + username);
    }

    public void registerSeller(String username, String email, double initialBalance) {
        Seller seller = new Seller(username, email);
        users.put(username, seller);
        financialService.addUser(username, initialBalance);
        productService.registerSeller(username);
        orderService.registerUser(username);
        System.out.println("Penjual baru terdaftar: " + username);
    }

    public void registerAdmin(String username, String email) {
        Admin admin = new Admin(username, email);
        users.put(username, admin);
        System.out.println("Admin baru terdaftar: " + username);
    }

    public User getUserByUsername(String username) {
        return users.get(username);
    }

    // Customer operations
```

```
public void buyProduct(String customerUsername, String sellerUsername, String product) {
    // Verify product availability
    if (productService.isProductAvailable(sellerUsername, product)) {
        double price = productService.getProductPrice(sellerUsername, product);
        // Process payment
        if (financialService.hasEnoughBalance(customerUsername, price)) {
            financialService.processPayment(customerUsername, sellerUsername, price);
            // Create order
            String orderId = orderService.createOrder(customerUsername, sellerUsername, product, price);
            // Send notifications
            notificationManager.sendNotification(customerUsername, "Pesanan " + orderId + " berhasil dibuat");
            notificationManager.sendNotification(sellerUsername, "Anda menerima pesanan baru: " + orderId);
            // Add loyalty points
            loyaltyService.addPoints(customerUsername, (int)(price / 10));
            // Track activity
            analyticsService.trackUserActivity(customerUsername, "Membeli produk: " + product);
        } else {
            notificationManager.sendNotification(customerUsername, "Saldo tidak mencukupi");
        }
    } else {
        notificationManager.sendNotification(customerUsername, "Produk tidak tersedia");
    }
}

public void submitProductReview(String customerUsername, String product, String content, int rating) {
    reviewService.submitReview(customerUsername, product, content, rating);
    analyticsService.trackUserActivity(customerUsername, "Mengirim ulasan produk");
}

public void redeemLoyaltyPoints(String customerUsername, int points) {
    if (loyaltyService.redeemPoints(customerUsername, points)) {
        double discount = points / 10.0;
        discountService.setUserDiscount(customerUsername, discount);
        notificationManager.sendNotification(customerUsername,
                "Berhasil menukar " + points + " poin untuk diskon " + discount + "%");
    } else {
        notificationManager.sendNotification(customerUsername, "Poin tidak mencukupi");
    }
}

// Seller operations
```

```
public void addProduct(String sellerUsername, String product, double price) {
    productService.addProduct(sellerUsername, product, price);
    analyticsService.trackUserActivity(sellerUsername, "Menambahkan produk: " + product);
}

public void addUnreleasedProduct(String sellerUsername, String product) {
    productService.addUnreleasedProduct(sellerUsername, product);
}

public void manageOrders(String sellerUsername) {
    List<Order> orders = orderService.getSellerOrders(sellerUsername);
    System.out.println(sellerUsername + " mengelola " + orders.size() + " pesanan");
    analyticsService.trackUserActivity(sellerUsername, "Mengelola pesanan");
}

public void shipOrder(String sellerUsername, String orderId) {
    shippingService.shipOrder(sellerUsername, orderId);
    orderService.updateOrderStatus(orderId, ShippingStatus.SHIPPED);
    analyticsService.trackUserActivity(sellerUsername, "Mengirim pesanan: " + orderId);
}

// Admin operations
public void moderateReview(String adminUsername, Review review, boolean approved) {
    reviewService.moderateReview(adminUsername, review, approved);
    analyticsService.trackUserActivity(adminUsername, "Memoderasi ulasan");
}

public void monitorUser(String adminUsername, String username) {
    securityService.monitorUser(username);
    analyticsService.trackUserActivity(adminUsername, "Memantau pengguna: " + username);
}

public void checkForFraud(String adminUsername, String username, double amount) {
    boolean isSuspicious = securityService.detectFraud(username, amount);
    if (isSuspicious) {
        notificationManager.sendNotification(adminUsername, "Aktivitas mencurigakan terdeteksi: " + username);
    }
    analyticsService.trackUserActivity(adminUsername, "Memeriksa kecurangan");
}

public void adjustUserBalance(String adminUsername, String username, double amount) {
```

```
    financialService.adjustBalance(username, amount);
    notificationManager.sendNotification(username, "Saldo Anda disesuaikan sebesar $" + amount);
    analyticsService.trackUserActivity(adminUsername, "Menyesuaikan saldo pengguna: " + username);
}

public List<String> getApprovedUnreleasedProducts(String adminUsername, String sellerUsername) {
    return productService.getApprovedUnreleasedProducts(sellerUsername, adminUsername);
}

public void generateAnalyticsReport(String adminUsername) {
    analyticsService.generateReport(adminUsername);
}

public static void main(String[] args) {
    ECommerceSystem system = new ECommerceSystem();

    // Register users
    system.registerAdmin("Charlie", "charlie@admin.com");
    system.registerCustomer("Alice", "alice@mail.com");
    system.registerSeller("Bob", "bob@store.com", 1000);

    // Fund customer account
    system.adjustUserBalance("Charlie", "Alice", 2000);

    // Customer actions
    User alice = system.getUserByUsername("Alice");
    alice.login();
    system.buyProduct("Alice", "Bob", "Laptop");
    system.submitProductReview("Alice", "Laptop", "Produk luar biasa!", 5);
    system.redeemLoyaltyPoints("Alice", 50);
    alice.logout();

    // Seller actions
    User bob = system.getUserByUsername("Bob");
    bob.login();
    system.addProduct("Bob", "Smartphone", 500);
    system.addProduct("Bob", "Laptop", 1000);
    system.addUnreleasedProduct("Bob", "Laptop Gaming");
    system.addUnreleasedProduct("Bob", "Smartphone 5G");
    system.manageOrders("Bob");
    system.shipOrder("Bob", "ORD1000");
```

```
        bob.logout();

        // Admin actions
        User charlie = system.getUserByUsername("Charlie");
        charlie.login();
        Review pendingReview = system.reviewService.getPendingReviews().get(0);
        system.moderateReview("Charlie", pendingReview, true);
        system.monitorUser("Charlie", "Alice");
        system.checkForFraud("Charlie", "Alice", 3000);
        system.getApprovedUnreleasedProducts("Charlie", "Bob");
        system.generateAnalyticsReport("Charlie");
        charlie.logout();
    }
}
```

BAB VI
LINK GITHUB

https://github.com/isanzzi/CBO-and-LCOM