LAPORAN PRAKTIKUM 9 ANALISIS PERANCANGAN PERANGKAT LUNAK

Analisis & Refaktorisasi Cohesion dan Coupling

Tugas

Disusun untuk memenuhi tugas APPL Praktek

Oleh :

Ahmad Fatan Haidar	231524034
Daffa Muzhaffar Fakhruddin	231524038
Muhammad Adhyaksa Fadillah	231524051
Muhammad Samudera Bagja	231524058
Nesta Rizkia Saputra	231524060

PROGRAM STUDI D4-TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI BANDUNG
2024

DAFTAR ISI

PENDAHULUAN

Langkah 1 – Analisis

- Jenis cohesion yang terjadi pada UtilityModule.java adalah **low cohesion**
  - Lebih spesifik, kemungkinan jenis cohesionnya adalah *logical cohesion* atau bahkan mendekati coincidental cohesion. Alasannya adalah karena modul ini menggabungkan fungsi-fungsi yang secara logika mungkin terkait dalam alur aplikasi (memproses pinjaman lalu mengirim email pemberitahuan), namun fungsi-fungsi tersebut memiliki tanggung jawab yang berbeda :
    - mengirim email,
    - memproses logika bisnis pinjaman dan,
    - mencatat aktivitas ke dalam log

    High cohesion berarti modul memiliki tanggung jawab yang jelas dan spesifik, di mana semua elemen dalam modul mendukung satu tujuan utama, yang tidak dipenuhi oleh UtilityModule yang ada pada tugas yang diberikan.
- Jenis coupling antar metode di modul tersebut adalah **tight coupling**
  - Metode processLoanRequest secara langsung memanggil metode *sendEmail.* Ini menciptakan ketergantungan langsung antar implementasi kedua metode2 Jika ada perubahan pada *sendEmail, processLoanRequest* mungkin perlu disesuaikan.
  - Semua metode (*sendEmail, processLoanRequest, logActivity*) berbagi dan bergantung pada variabel global globalStatus. Hal tersebut adalah contoh **common coupling**, yang merupakan bentuk coupling yang sangat ketat. Dimana Perubahan pada variabel global dapat mempengaruhi perilaku semua metode yang menggunakannya, dan sulit untuk melacak dari mana perubahan tersebut berasal.
- Konsekuensi jika modul seperti ini terus digunakan dalam sistem skala besar adalah:
  - Sulit Pemeliharaan (Low Maintainability): Perubahan pada satu bagian modul dapat secara tidak terduga mempengaruhi bagian lain karena ketergantungan yang tinggi (tight coupling) dan berbagi status global. Hal ini meningkatkan risiko bug dan mempersulit proses pemeliharaan. Tujuan utama software design adalah menciptakan sistem yang mudah dipahami, diubah, dan dipelihara. Modul dengan low cohesion dan high coupling bertentangan dengan tujuan ini.
  - Sulit Pengujian (Difficult Testing): Karena fungsi-fungsi dalam modul saling bergantung dan berbagi status, pengujian unit menjadi lebih rumit. Untuk menguji satu fungsi, kita mungkin perlu menyiapkan kondisi dan status dari fungsi atau variabel global lain yang terkait. Hal ini mengurangi efektivitas pengujian dan meningkatkan kemungkinan adanya bug yang tidak terdeteksi.
  - Kurang Fleksibel dan Sulit Digunakan Kembali (Low Reusability): Modul yang melakukan banyak hal yang tidak terkait sulit untuk

digunakan kembali di bagian lain sistem atau dalam proyek lain yang hanya membutuhkan sebagian dari fungsionalitasnya.

- Peningkatan Kompleksitas: Semakin banyak tanggung jawab yang digabungkan dalam satu modul dengan ketergantungan yang tinggi, semakin sulit untuk memahami dan mengelola kode secara keseluruhan.

Langkah 2 – Refaktorisasi Desain

`UtilityModule` dipecah menjadi 3 kelas dengan tanggung jawab spesifik:

1.1.1 Desain Baru:

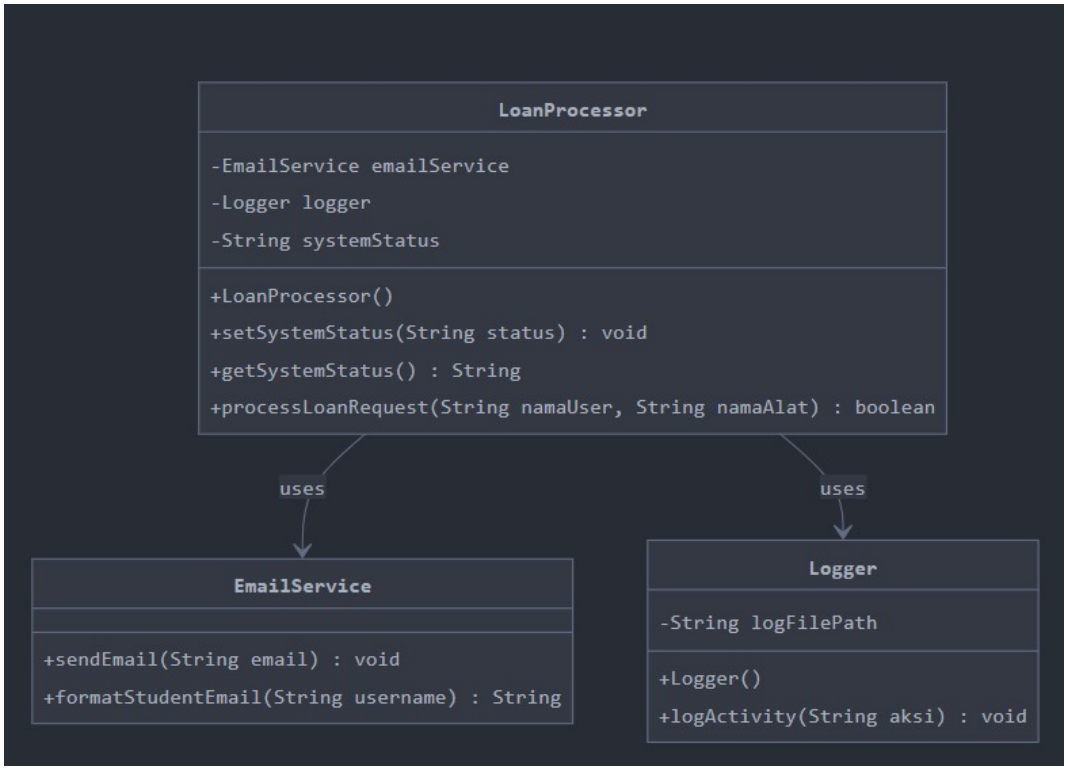
```
/* EmailService - Kelas dengan tanggung jawab khusus untuk menangani email,
Menerapkan prinsip high cohesion dengan fokus pada satu tujuan
*/
public class EmailService {



   public void sendEmail(String email) {
      System.out.println("Sending email to: " + email);
   }


   public String formatStudentEmail(String username) {
      return username + "@student.pnp.ac.id";
   }
}
```

```
/*
 Logger - Kelas dengan tanggung jawab khusus untuk logging
 Menerapkan prinsip high cohesion dan encapsulation untuk operasi logging
*/
public class Logger {
   private String logFilePath;

   public Logger() {
      this.logFilePath = "log.txt";
   }

   public void logActivity(String aksi) {
      try {
```

```
            java.io.FileWriter writer = new java.io.FileWriter(logFilePath, true);
            writer.write(aksi + "\n");
            writer.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
/*
  LoanProcessor - Kelas utama untuk memproses permintaan peminjaman
  Menerapkan prinsip dependency injection untuk mengurangi coupling
 */
public class LoanProcessor {
    private EmailService emailService;
    private Logger logger;
    private String systemStatus;

    /**
     * Konstruktor yang menginisialisasi processor dengan dependencies-nya
     */
    public LoanProcessor() {
        this.emailService = new EmailService();
        this.logger = new Logger();
        this.systemStatus = "AKTIF";
    }

    public void setSystemStatus(String status) {
        this.systemStatus = status;
        logger.logActivity("System status changed to: " + status);
    }

    /**
     * Mendapatkan status sistem saat ini
    public String getSystemStatus() {
        return systemStatus;
    }

    public boolean processLoanRequest(String namaUser, String namaAlat) {
        if (systemStatus.equals("AKTIF")) {
            // Mencatat aktivitas peminjaman
            String logMessage = "Alat " + namaAlat + " dipinjam oleh " + namaUser;
            System.out.println(logMessage);
            logger.logActivity(logMessage);
```

```
        // Mengirim email pemberitahuan
        String emailAddress = emailService.formatStudentEmail(namaUser);
        emailService.sendEmail(emailAddress);

        return true;
      } else {
        String message = "Peminjaman gagal: Sistem sedang offline";
        System.out.println(message);
        logger.logActivity(message);
        return false;
      }
    }
}
```



Gambar 1 Class Diagram Setelah di refactor

1.1.2 Penjelasan Refaktorisasi:

Refaktorisasi ini membagi tanggung jawab menjadi tiga kelas yang berbeda, masing-masing dengan satu tanggung jawab utama yang koheren:

1. **EmailService**: Menangani semua yang terkait dengan email

a. Memiliki **functional cohesion** karena semua metodenya terkait dengan fungsionalitas email

b. Metode sendEmail() dan formatStudentEmail() bekerja sama untuk menyelesaikan tugas terkait email

2. **Logger**: Bertanggung jawab hanya untuk mencatat aktivitas sistem

a. Memiliki **functional cohesion** karena fokus pada satu fungsi (logging)

b. Menyembunyikan detail implementasi logging dengan enkapsulasi

3. **LoanProcessor**: Menangani logika peminjaman alat

a. Memiliki **logical cohesion** karena metode-metodenya terkait dengan pemrosesan peminjaman

b. Mengelola status sistem secara terenkapsulasi (private)

Langkah 3 – Review Singkat

1.1.3 Perbaikan yang dilakukan untuk mengurangi coupling

1. **Menghilangkan common coupling**:

a. Mengganti variabel statis global globalStatus dengan variabel instance systemStatus yang dienkapsulasi

b. Menyediakan getter/setter untuk mengakses status sistem

2. **Mengurangi content coupling**:

a. LoanProcessor menggunakan objek EmailService dan Logger dengan tanggung jawab yang jelas

b. Komunikasi antar objek melalui metode publik yang terdefinisi dengan jelas

3. **Mengurangi ketergantungan langsung**:

a. LoanProcessor menciptakan objek dependensi-nya sendiri, namun secara struktur sudah lebih baik

1.1.4 Perbandingan dengan Kode Awal

1. **Peningkatan Cohesion**:

a. Dari coincidental cohesion (kelas berisi fungsi yang tidak terkait) menjadi functional cohesion (kelas dengan satu fungsi utama)

   b. Setiap kelas memiliki tanggung jawab yang jelas dan fokus sesuai prinsip Single Responsibility

2. **Pengurangan Coupling**:

   a. Dari tight coupling (variabel global dan pemanggilan langsung) menjadi loose coupling (komunikasi melalui metode publik)

   b. Menghilangkan ketergantungan pada variabel global

3. **Enkapsulasi yang Lebih Baik**:

   a. Status sistem sekarang dienkapsulasi dan hanya dapat diakses melalui metode publik

   b. Implementasi internal setiap kelas disembunyikan dari kelas lain