**LAPORAN TUGAS WORKSHOP**

“Tugas Refactoring dan Clean Code Practice”

Mata Kuliah Aplikasi Workshop Rekayasa Ulang Kode



Nama dosen pengampu:

Adam Shidqul Aziz S.ST., M.T

Dikerjakan oleh:

Nama : Kansa Adeneva

NRP : 3123600009

Kelas : 2 D4 IT A

1. Refactoring
   1. Code bad

public class codeconventionbad(){

public static void main(String argS[])

{

System.out.println("PROGRAM HITUNG gaji===>");

employeE\_data e=new employeE\_data();

e.procESScalc("budi",40,50000);

e.procESScalc("ANI",38,45000);

managEr m=new managEr("toni",60000,45);

m.HITUNGGaJI();

}}

public class employeE\_data{

void procESScalc(String NAMA,int JAM,int u){

int hasil=JAM\*u;

System.out.println("nama:"+NAMA+" gaji="+hasil);}}

class managEr{

String nm;

int SALARY;

int JAMkerja;

public managEr(String nm,int SALARY,int JamKerja){

this.nm=nm;this.SALARY=SALARY;this.JAMkerja=JamKerja;}

public void HITUNGGaJI(){int g=this.SALARY\*this.JAMkerja;

System.out.println("ManagerName:"+nm+"->"+g);}}

* 1. Code clean (sudah diperbaiki)

public class CodeConventionClean {

    public static void main(String[] args) {

        System.out.println("=== Program Hitung Gaji ===");

        Employee employee1 = new Employee();

        employee1.calculateSalary("Budi", 40, 50000);

        Employee employee2 = new Employee();

        employee2.calculateSalary("Ani", 38, 45000);

        Manager manager = new Manager("Toni", 60000, 45);

        manager.calculateSalary();

    }

}

class Employee {

    void calculateSalary(String name, int workHours, int hourlyRate) {

        int totalSalary = workHours \* hourlyRate;

        System.out.println("Nama: " + name + " | Gaji: " + totalSalary);

    }

}

class Manager {

    private String name;

    private int salaryPerHour;

    private int workHours;

    public Manager(String name, int salaryPerHour, int workHours) {

        this.name = name;

        this.salaryPerHour = salaryPerHour;

        this.workHours = workHours;

    }

    public void calculateSalary() {

        int totalSalary = salaryPerHour \* workHours;

        System.out.println("Manager Name: " + name + " | Gaji: " + totalSalary);

    }

}

* 1. Analisa
     1. Nama class tidak sesuai konvensi
        + codeconventionbad , employeE\_data, managEr → tidak pakai PascalCase.
        + **Perbaikan:** diganti menjadi CodeConventionClean, Employee, Manager.
     2. Method & variabel tidak konsisten penamaannya
        + procESScalc, HITUNGGaJI → penulisan random huruf besar/kecil.
        + Variabel NAMA, JAM, u, nm, SALARY, JAMkerja → membingungkan dan tidak jelas artinya.
        + **Perbaikan:** diganti menjadi calculateSalary, name, workHours, hourlyRate, salaryPerHour.
     3. Output tidak rapi / tidak konsisten
        + "PROGRAM HITUNG gaji===>" → tidak konsisten kapitalisasi dan formatting.
        + Output "nama:"+NAMA+" gaji="+hasil; → kurang informatif.
        + **Perbaikan:** "=== Program Hitung Gaji ===" dan "Nama: " + name + " | Gaji: " + totalSalary.
     4. Tidak ada access modifier (encapsulation)
        + Field String nm, int SALARY, int JAMkerja → tidak private.
        + Perbaikan: pakai private di class Manager
     5. Nama constructor parameter tidak konsisten
        + String nm, int SALARY, int JamKerja → ada huruf random besar kecil.
        + Perbaikan: String name, int salaryPerHour, int workHours.
     6. Class Manager method tidak deskriptif
        + Method HITUNGGaJI() → semua capslock, tidak sesuai Java convention.
        + Perbaikan: jadi calculateSalary()
     7. Penulisan kurung kurawal & indentasi
        + Awalnya { sering di baris berbeda & indentasi berantakan.
        + Perbaikan: semua { dibuka di baris yg sama, indentasi konsistensi 4 spasi
     8. Hasil Perbaikan
        + **Class** pakai PascalCase: CodeConventionClean, Employee, Manager.
        + **Method & variabel** pakai camelCase: calculateSalary, workHours, hourlyRate.
        + **Konstanta** tidak ada di studi kasus ini, tapi kalau ada wajib UPPER\_SNAKE\_CASE.
        + **Encapsulation**: field di Manager dibuat private.
        + **Output** lebih rapi, mudah dibaca: "Nama: Budi | Gaji: 2000000".
        + **Kode** jadi lebih konsisten, mudah dipahami, dan bisa dikembangkan.

1. Clean Code Practice

public class PenghitungNilai {

    public static void main(String[] args) {

        String namaMahasiswa = "Budi";

        int nilaiTugas = 85;

        int nilaiUts = 78;

        int nilaiUas = 90;

        double rataRata = (nilaiTugas + nilaiUts + nilaiUas) / 3.0;

        System.out.println("Nama Mahasiswa: " + namaMahasiswa);

        System.out.println("Nilai Tugas   : " + nilaiTugas);

        System.out.println("Nilai UTS     : " + nilaiUts);

        System.out.println("Nilai UAS     : " + nilaiUas);

        System.out.println("Rata-rata     : " + rataRata);

        String grade;

        if (rataRata >= 85) {

            grade = "A";

        } else if (rataRata >= 70) {

            grade = "B";

        } else if (rataRata >= 55) {

            grade = "C";

        } else if (rataRata >= 40) {

            grade = "D";

        } else {

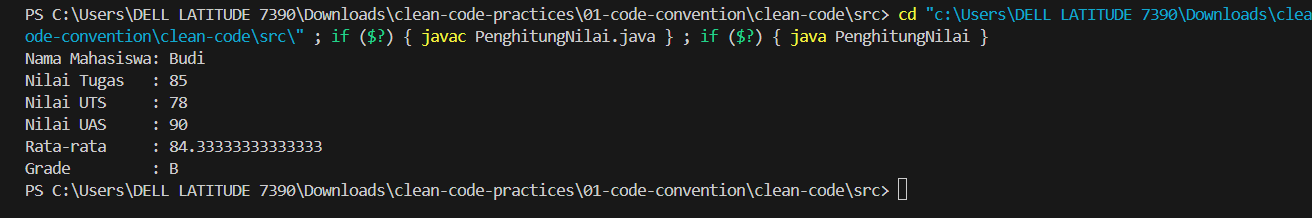
            grade = "E";

        }

        System.out.println("Grade         : " + grade);

    }

}

Output:  


1. Analisa
   1. Struktur kelas
      1. Nama kelas ‘PenghitungNilai’ sudah menggunakan PascalCase dan nama jelas mewakili fungsi
   2. Nama variabel
      1. Variabel yg digunakan ‘nilaiTugas’, ‘nilaiUts’, ‘nilaiUas’, ‘nilaiAkhir’ menggunakan camelCase dan jelas mempresenasikan tujuan agar mudah dipahami
   3. Fungsi & metode
      1. Metode ‘hitungNilaiAkhir’ menerima parameter nilai tugas, UTS, dan UAS lalu menghitung nilai akhir. Penamaan metode menggunakan camelCase, memiliki satu tanggug jawab utama (single responsibility principle) tidak duplikasi kode