

**LAPORAN PRAKTIKUM  
TEKNIK PEMROGRAMAN**

**COUPLING COHESION**

**MINGGU KE-7**



**POLBAN**

**NAMA: FAUZI ISMAIL**

**NIM: 241524042**

**KELAS: D4-1B**

**PROGRAM STUDI SARJANA TERAPAN**

**TEKNIK INFORMATIKA**

**POLITEKNIK NEGERI BANDUNG**

**2025**

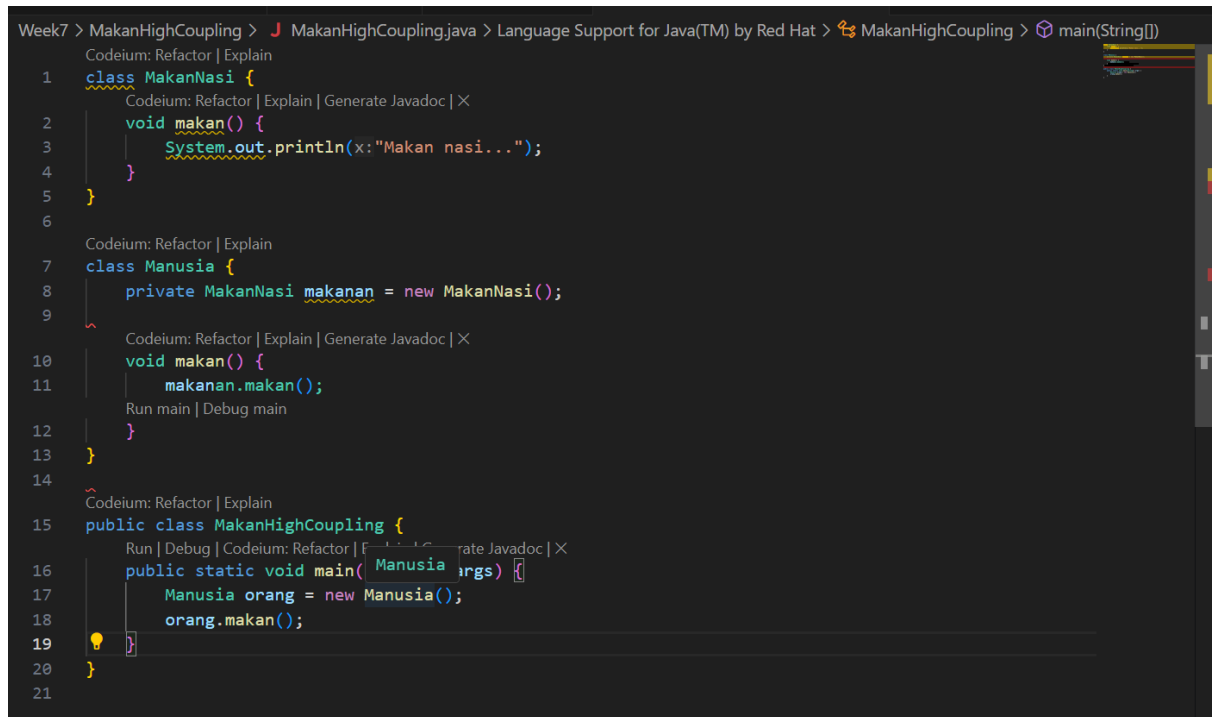
# DAFTAR ISI

DAFTAR ISI.....	2
<a href="https://github.com/mailvlous/teknikPemrograman/tree/main/Week6">https://github.com/mailvlous/teknikPemrograman/tree/main/Week6</a> .....	3
1. Coupling Between Objects.....	3
1. High Coupling.....	3
2. Low Coupling .....	4
2. Lack of Cohesion in Methods .....	5

<https://github.com/mailvlous/teknikPemrograman/tree/main/Week7>

## 1. Coupling Between Objects

### 1. High Coupling



```
Week7 > MekanHighCoupling > MekanHighCoupling.java > Language Support for Java(TM) by Red Hat > MekanHighCoupling > main(String[])
Codeium: Refactor | Explain
1  class MekanNasi {
2      void makan() {
3          System.out.println(x:"Makan nasi...");
4      }
5  }
6
Codeium: Refactor | Explain
7  class Manusia {
8      private MekanNasi makanan = new MekanNasi();
9
10     void makan() {
11         makanan.makan();
12     }
13 }
14
Codeium: Refactor | Explain
15 public class MekanHighCoupling {
16     public static void main( Manusia args) {
17         Manusia orang = new Manusia();
18         orang.makan();
19     }
20 }
21
```

Ini contoh termasuk high coupling karena:

1. Manusia tidak dapat makan jenis makanan yang berbeda. Manusia hanya dapat memanggil makan lalu makan nasi.
2. Class Manusia terlalu ketergantungan terhadap Class MekanNasi. Misal manusia ingin menambahkan function lain maka Class MekanNasi juga harus ikut berubah.

- Manusia mengakses MekanNasi = 1
- MekanHighCoupling mengakses Manusia = 1
- $1+1 = 2$

## 2. Low Coupling

```
Week7 > MakanLowCoupling > J MakanLowCoupling.java > Language Support for Java(TM) by Red Hat > MakanLowCoupling > main(String[])

Codeium: Refactor | Explain
1 interface Makanan {
2     void makan();
3 }
4
Codeium: Refactor | Explain
5 class Nasi implements Makanan {
6     Codeium: Refactor | Explain | Generate Javadoc | X
7     public void makan() {
8         System.out.println(x:"Makan nasi...");
9     }
10
Codeium: Refactor | Explain
11 class Roti implements Makanan {
12     Codeium: Refactor | Explain | Generate Javadoc | X
13     public void makan() {
14         System.out.println(x:"Makan roti...");
15     }
16
Codeium: Refactor | Explain
17 class Manusia {
18     private Makanan makanan;
19
20     public Manusia(Makanan makanan) {
21         this.makanan = makanan;
22     }
23
24     Codeium: Refactor | Explain | Generate Javadoc | X
25     void makan() {
26         makanan.makan();
27     }
28
Codeium: Refactor | Explain
29 public class MakanLowCoupling {
30     Run | Debug | Run main | Debug main | Codeium: Refactor | Explain | Generate Javadoc | X
31     public static void main(String[] args) {
32         Manusia orang1 = new Manusia(new Nasi());
33         Manusia orang2 = new Manusia(new Roti());
34         orang1.makan();
35         orang2.makan();
36     }
37 }
```

Ini kasusnya sama seperti yang diatas namun kali ini termasuk low coupling, karena:

1. Menggunakan interface sehingga Class Manusia tidak bergantung terhadap Class MakanNasi
2. Class Manusia hanya bergantung pada Makanan, bukan secara langsung terhadap MakanNasi

- MakanNasi mengakses interface Makanan = 1
- Manusia mengakses interface Makanan = 1
- MakanLowCoupling mengakses Manusia = 1
- $1+1+1=3$

## 2. Lack of Cohesion in Methods

```
Week7 > MakanHighCoupling > J MakanHighCoupling.java > ...
Codeium: Refactor | Explain
1  class MakanNasi {
2      Codeium: Refactor | Explain | Generate Javadoc | X
3      void makan() {
4          System.out.println(x:"Makan nasi...");
5      }
6
7  class Manusia {
8      private MakanNasi makanan = new MakanNasi();
9      private String nama;
10
11     Codeium: Refactor | Explain | Generate Javadoc | X
12     void makan() {
13         makanan.makan();
14     }
15
16     Codeium: Refactor | Explain | Generate Javadoc | X
17     void setName(String nama) {
18         this.nama = nama;
19     }
20
21     Codeium: Refactor | Explain | Generate Javadoc | X
22     void tampilkanNama() {
23         System.out.println("Nama: " + nama);
24     }
25 }

Week7 > MakanHighCoupling > J MakanHighCoupling.java > ...
7  class Manusia {
19     void tampilkanNama() {
21     }
22 }
23
24 Codeium: Refactor | Explain
25 public class MakanHighCoupling {
26     Run | Debug | Run main | Debug main | Codeium: Refactor | Explain | Generate Javadoc | X
27     public static void main(String[] args) {
28         Manusia orang = new Manusia();
29         orang.makan();
30     }
31 }
```

Mari kita hitung pada kasus ini:

1.

- Pada function makan(), makan() menggunakan class Makanan
- setName menggunakan atribut nama
- tampilkanNama menggunakan atribut nama

2.

Pasangan yang tidak berbagi atribut (P):

- Makan() tidak berbagi atribut dengan setName()
- Makan() tidak berbagi atribut dengan tampilkanNama()

Pasangan yang berbagi atribut (Q):

- `setNama()` dan `tampilkanNama()` berbagi atribut nama

$$LCOM = P - Q = 2 - 1 = 1$$

Low Cohesion