VERSI 0.1

AGUSTUS 2025

PEMROGRAMAN LANJUT

MODUL 3 - MODERN PROGRAMMING ENVIRONMENT AND DOCUMENTATION STYLE

DISUSUN OLEH:

Ir. Wildan Suharso, M.Kom.

Muhammad Ega Faiz Fadlillah

M. Ramadhan Titan Dwi .C

TIM LABORATORIUM INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

PENDAHULUAN

TUJUAN

1. Memahami dan mengoperasikan Integrated Development Environment (IDE) untuk pengembangan perangkat lunak secara efektif..
2. Menggunakan alat pengelolaan dependensi seperti Maven atau Gradle untuk mengelola pustaka dan komponen eksternal.
3. Mengimplementasikan pengendalian versi menggunakan **Git** untuk memanage kode dan kolaborasi tim.
4. Memahami konsep dasar Continuous Integration (CI) dan Continuous Deployment (CD) untuk memastikan kualitas kode dan penerapan perubahan perangkat lunak yang tepat.
5. Menyusun dokumentasi kode yang efektif menggunakan JavaDoc dan README, serta mengimplementasikan komentar yang jelas dan terstruktur di dalam kode.

TARGET MODUL

1. Membuat proyek Java lengkap dengan dependensi menggunakan Maven/Gradle.
2. Menggunakan Git untuk pengendalian versi dan mengelola pengembangan fitur melalui branching.
3. Menyusun dokumentasi yang sesuai, baik melalui JavaDoc untuk kode sumber maupun file README untuk proyek.

PERSIAPAN

1. Java Development Kit (JDK) - Untuk pengembangan aplikasi berbasis Java.
2. IDE - Disarankan menggunakan IDE seperti IntelliJ IDEA, Visual Studio Code.
3. Git.
4. Maven atau Gradle - Untuk manajemen dependensi proyek.

KEYWORDS

IDE, Git, JavaDoc, Version Control. README, Maven, Debugging

TABLE OF CONTENTS

TEORI

A. *Modern Programming Environment*

Integrated Development Environment **(IDE)** adalah perangkat lunak berbasis antarmuka grafis pengguna **(GUI)** yang digunakan untuk menulis dan menjalankan kode pemrograman. IDE seperti IntelliJ IDEA dan Eclipse, yang akan kita gunakan dalam mata kuliah pemrograman lanjut ini, menyediakan berbagai fitur yang mendukung produktivitas pengembang, seperti debugging, autocompletion, dan integrasi dengan sistem kontrol versi seperti Git. Dalam praktikum ini, mahasiswa akan mempelajari cara membuat proyek, menambahkan dependensi menggunakan Maven atau Gradle, serta mengintegrasikan perubahan kode ke dalam repositori Git. Selain itu, mahasiswa juga akan memahami konsep Continuous Integration **(CI)** dan Continuous Deployment **(CD)** untuk memastikan bahwa setiap perubahan kode diuji dan diterapkan dengan benar dalam siklus pengembangan perangkat lunak.

1. *Konsep IDE*

Selain berfungsi untuk menulis dan menjalankan kode pemrograman, **IDE** juga bertujuan untuk meningkatkan efisiensi pengembang dengan menggabungkan berbagai fitur seperti pengeditan, pembangunan, pengujian, dan pengemasan perangkat lunak dalam satu aplikasi yang mudah digunakan. IDE menyediakan berbagai alat yang diperlukan oleh programmer, sehingga mempermudah dan mempercepat proses pengembangan perangkat lunak.

2. *Fitur IDE*

Berikut ini adalah fitur-fitur yang dapat digunakan programmer pada IDE Intellij IDEA:

- **Debugging**

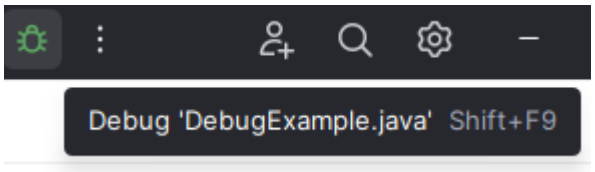
Debugging adalah proses untuk menemukan dan memperbaiki kesalahan dalam sebuah program. Terdapat berbagai jenis kesalahan yang dapat diatasi melalui debugging, seperti kesalahan sintaksis. Kesalahan sederhana sering kali mudah ditemukan dengan memeriksa call stack, yang membantu mengidentifikasi lokasi kesalahan. Namun, ada juga kesalahan yang lebih kompleks yang membutuhkan waktu lebih lama untuk dideteksi dan diperbaiki. Di sinilah debugging sangat berguna. Proses debugging dilakukan dengan menghentikan eksekusi program sementara, lalu menganalisis kondisi program secara lebih mendalam, termasuk memeriksa nilai variabel dan perubahan yang terjadi pada setiap baris kode.

Cara Menggunakan Debugging:
1) Letakkan breakpoint (titik henti) pada baris kode yang ingin kamu periksa.

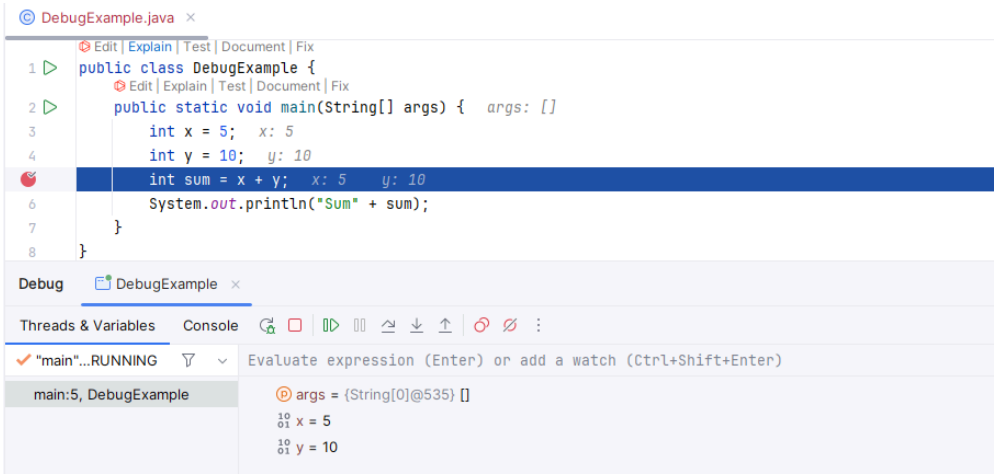
```
1  public class DebugExample {
       Edit | Explain | Test | Document | Fix
2      public static void main(String[] args) {
3          int x = 5;
4          int y = 10;
5          int sum = x + y;
6          System.out.println("Sum" + sum);
7      }
8  }
9
```

caranya, dengan klik Gutter (klik area di sebelah kiri nomor). Klik di nomor baris yang kamu pilih. Kamu akan melihat titik merah muncul, menandakan bahwa breakpoint telah ditambahkan\

2) Jalankan program dalam mode debug dengan mengklik tombol "Debug" atau menggunakan pintasan keyboard.



3) Kode akan berhenti di breakpoint yang ditentukan. Kamu dapat melihat nilai variabel, menggerakkan eksekusi langkah demi langkah, dan melacak jejak pemanggilan.

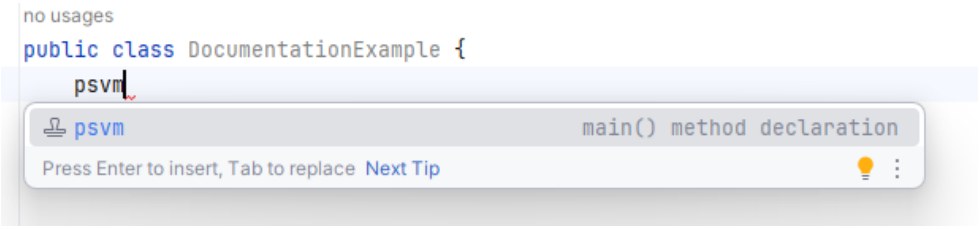


- **Autocomplete**

Autocomplete (pengisian otomatis) adalah fitur yang mempermudah penulisan kode dengan memberikan saran otomatis untuk nama variabel, metode, kelas, dan elemen kode lainnya saat kita mengetik. Fitur ini sangat berguna untuk meningkatkan produktivitas pengembang serta mengurangi kemungkinan kesalahan penulisan kode.

Contoh Cara Menggunakan Autocomplete :

1) Mulai mengetik kode atau kata kunci.
Misalkan disini kita akan mencoba untuk menulis `public static void main` , cara mudah agar kita tidak perlu menulis panjang, maka kita bisa menuliskan psvmt saja
2) IntelliJ IDEA akan menampilkan pilihan autocomplete di bawah input Anda.



3) Pilih pilihan yang diinginkan dengan menggunakan tombol panah atau klik mouse. Dan
kode yang kita inginkan akan langsung tertulis lengkap tanpa kita mengetik panjang

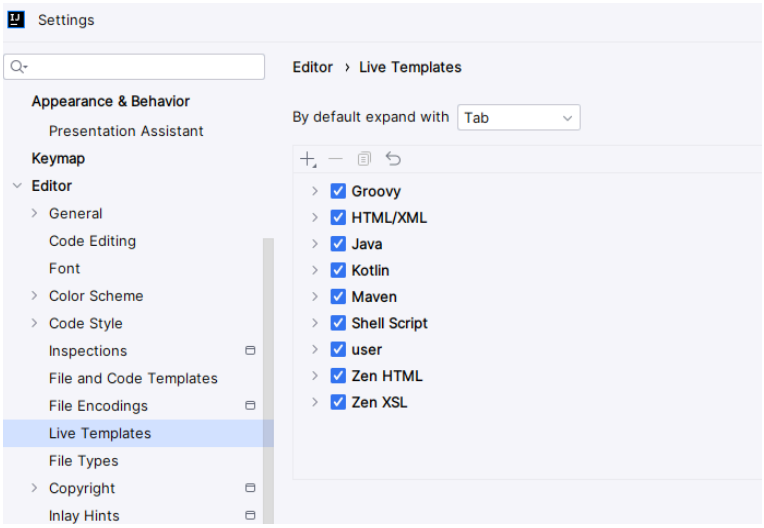
```
public class DocumentationExample {
    public static void main(String[] args) {

    }
}
```

Berikut ini adalah daftar untuk keyword autocomplete

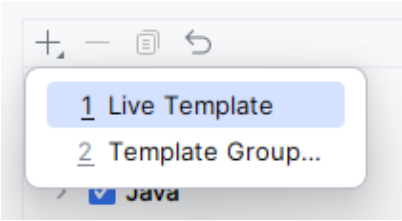
Keyword Autocomplete	Deskripsi
sout	Menyederhanakan penulisan System.out.println() dengan mengetik "sout" diikuti dengan tombol Tab.

psvm	Membuat metode public static void main(String[] args) dengan mengetik "psvm" diikuti dengan tombol Tab.
fori	Membuat perulangan for dengan mengisi variabel dan batasan indeks dengan mengetik "fori" diikuti dengan tombol Tab.
iter	Membuat perulangan for-each untuk mengiterasi elemen dalam koleksi dengan mengetik "iter" diikuti dengan tombol Tab.
ifn	Membuat pernyataan if (variable == null) untuk memeriksa apakah variabel null dengan mengetik "ifn" diikuti dengan tombol Tab.
psf	Membuat konstanta public static final dengan mengetik "psf" diikuti dengan tombol Tab.
main	Membuat metode public static void main(String[] args) dengan mengetik "main" diikuti dengan tombol Tab.
cast	Melakukan casting tipe data dengan mengetik "cast" diikuti dengan tombol Tab.
Ctrl + space	Menggunakan kombinasi tombol ini akan menampilkan autocomplete dengan pilihan lengkap untuk kode yang Anda ketikkan.
Ctrl + shift + space	Menggunakan kombinasi tombol ini akan menampilkan autocomplete berdasarkan konteks, memberikan saran yang lebih akurat sesuai dengan tempat di mana Anda sedang menulis kode.

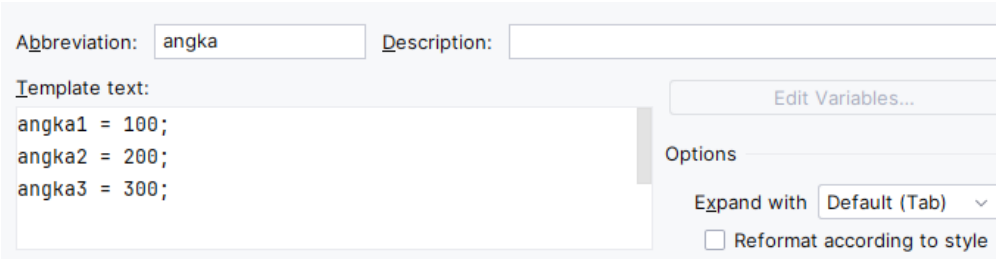
Cara membuat custom live code template:
Ctrl+Alt+S atau dari File > Settings. Setelah itu akan muncul pop up berikut. Pilih Editor dan klik Live Templates.



Klik tanda + dan pilih Live Template



Rename <abbreviation> dengan nama yang mudah diingat lalu isi bagian template text dengan code yang ingin kalian jadikan sebagai templates.

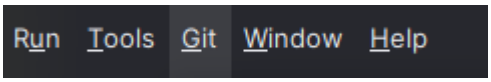


- **Integrasi GIT**

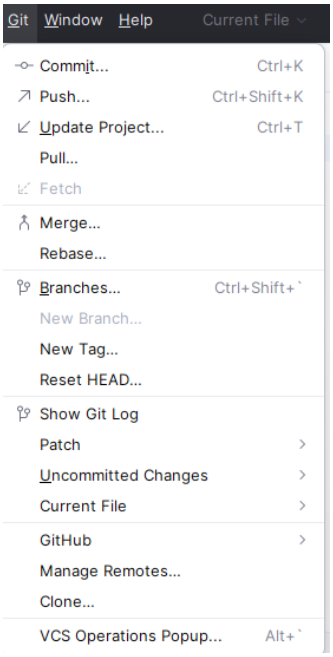
Integrasi Git dalam IntelliJ IDEA memungkinkan pengelolaan kode sumber

langsung melalui antarmuka pengguna IDE, tanpa perlu berpindah ke terminal atau aplikasi lain. Anda dapat melakukan berbagai operasi pengendalian versi, seperti commit, push, pull, dan lainnya, langsung dari dalam IntelliJ IDEA.

Cara Menggunakan Integrasi Git:
1) Buka proyek Anda dalam IntelliJ IDEA.
2) Di panel atas, Anda akan melihat tab "Version Control" yang menghubungkan ke repositori Git.



3) Gunakan fitur-fitur seperti commit, push, pull, dan lainnya melalui antarmuka pengguna
yang disediakan oleh IntelliJ IDEA.



Contoh :
Untuk melakukan commit: Klik kanan pada file yang ingin Anda commit, pilih "Git" > "Commit
File", lalu masukkan pesan commit dan klik "Commit".
Untuk melakukan push: Klik kanan pada proyek, pilih "Git" > "Repository" > "Push", lalu pilih
cabang yang ingin Anda dorong ke repositori.

- **Penggunaan Dependency Management (Maven atau Gradle)**

Dependency Management adalah proses untuk mengelola pustaka eksternal yang dibutuhkan oleh proyek Anda. Alat seperti Maven atau Gradle membantu mengotomatisasi proses pengunduhan, pengelolaan, dan integrasi dependensi ke dalam proyek. Hal ini memungkinkan Anda untuk lebih fokus pada pengembangan fitur dan fungsionalitas aplikasi, tanpa perlu repot mengelola dependensi secara manual.

Cara Menggunakan Dependency Management:

1. Menggunakan Maven:

   1) Buka proyek Anda di IntelliJ IDEA.

   2) Pastikan Anda memiliki file pom.xml di akar proyek Anda.

   3) Buka file pom.xml dan tambahkan dependensi di dalam tag <dependencies>.

```
<dependencies>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-lang3</artifactId>
        <version>3.12.0</version>
    </dependency>
</dependencies>
```

   4) IntelliJ IDEA akan mendeteksi perubahan dalam file pom.xml dan secara otomatis
   mengunduh dependensi yang didefinisikan.

   Menggunakan Gradle:

   1) Buka proyek Anda di IntelliJ IDEA.

   2) Pastikan Anda memiliki file build.gradle di akar proyek Anda.

   3) Buka file build.gradle dan tambahkan dependensi di dalam blok dependencies.

```
dependencies {
    implementation 'org.apache.commons:commons-lang3:3.12.0'
}
```

   4) IntelliJ IDEA akan mendeteksi perubahan dalam file build.gradle dan secara otomatis
   mengunduh dependensi yang didefinisikan.

- **Memahami Continuous Integration/Continuous Deployment (CI/CD) dalam Pengembangan Perangkat Lunak**

  Continuous Integration (CI) dan Continuous Deployment (CD) adalah metode dalam pengembangan perangkat lunak yang mengandalkan otomatisasi untuk pengujian, penggabungan kode, dan pengiriman perubahan ke lingkungan produksi. Pendekatan CI/CD ini berfungsi untuk menjaga kualitas kode, meminimalkan potensi kesalahan, serta memungkinkan proses pengembangan yang lebih cepat dan lebih aman.

  Cara Memahami CI/CD:

  1. Memahami Continuous Integration (CI): CI berfokus pada penggabungan kode secara berkala dari berbagai anggota tim, diikuti dengan otomatisasi pengujian unit dan integrasi untuk mendeteksi masalah sejak dini.

  2. Memahami Continuous Deployment (CD): CD melibatkan pengiriman otomatis perubahan kode yang telah lolos pengujian ke lingkungan produksi, dengan intervensi minimal atau tanpa perlu campur tangan manual.

  Penerapan CI/CD di IntelliJ IDEA:

  1. Dengan menggunakan alat CI/CD seperti Jenkins, Anda dapat mengonfigurasi otomatisasi pengujian dan pengiriman perubahan kode dari repositori ke lingkungan produksi.

  2. IntelliJ IDEA terintegrasi dengan alat CI/CD seperti Jenkins, yang memungkinkan Anda untuk mengelola dan mengontrol proses CI/CD langsung dari dalam IDE, mempermudah pengelolaan pengujian dan penerapan kode.

B. Documentation Style

Dokumentasi adalah proses menyusun penjelasan dan informasi yang berkaitan dengan bagian-bagian dari sebuah proyek perangkat lunak. Tujuan utamanya adalah untuk mempermudah pemahaman, pemeliharaan, dan penggunaan kode oleh pengembang lain atau pengembang yang akan bekerja dengan proyek tersebut di masa depan.

1. Jenis Dokumentasi

- **Kode Sumber**

  Kode sumber merujuk pada penjelasan singkat yang ditulis di samping atau di atas baris kode untuk menjelaskan fungsi atau tujuan dari kode tersebut.
  Contoh :

```java
public class DocumentationExample {
    Edit | Explain | Test | Document | Fix
    public static void main(String[] args) {
        String kalimat1 = "saya bapak Budi"; //kalimat 1
        String kalimat2 = "saya ibu Budi"; //kalimat 2
        System.out.println("Berikut : " + kalimat1 + " " + kalimat2); // menampilkan semua kalimat
    }
}
```

- **JavaDoc**

  JavaDoc adalah alat dokumentasi untuk bahasa pemrograman Java yang digunakan untuk menghasilkan dokumentasi yang terstruktur berdasarkan komentar dalam kode sumber Java. Dengan menggunakan tag seperti **@param**, **@return**, dan lainnya, JavaDoc memungkinkan pengembang untuk mendokumentasikan kelas, metode, dan elemen lainnya secara terstandarisasi, sehingga memudahkan pemahaman serta pemeliharaan kode.

  Cara Penggunaan JavaDoc :

  1. Tempatkan kursor di atas kelas, method, atau anggota lain yang ingin Anda dokumentasikan
  2. Ketika /** pada baris di atas elemen tersebut dan tekan enter. Intellij IDEA akan secara otomatis membuat template JavaDoc dan menempatkan kursor di tempat yang sesuai untuk mengisi komentar
  3. Isi detail seperti deskripsi method, parameter, return, dan pengecualian yang mungkin dibuang

  Contoh :

```java
public class Calculator {
    2 usages
    private int x;
    1 usage
    private int y;
    no usages
    private int sum;

    Edit | Explain | Test | Document | Fix
    /**
     * @param x nilai x
     * @param y nilai y
     */
    no usages
    public Calculator(int x, int y){
        this.x = x;
        this.y = y;
    }

    Edit | Explain | Test | Document | Fix
    /**
     * Mendapatkan nilai x
     * @return x
     */
    no usages
    public int getX() {
        return x;
    }
}
```

- **README**

  README adalah dokumen panduan yang umumnya berisi deskripsi tentang proyek, petunjuk instalasi, cara penggunaan, serta informasi mengenai kontribusi yang dapat dilakukan oleh pihak lain.

README.md

[MODUL 3]

Fitur Utama

- Fitur 1
- Fitur 2
- Fitur 3

Instalasi

Untuk menjalankan proyek ini secara lokal, ikuti langkah-langkah berikut:

1. **Clone repository**:

```
git clone https://github.com/username/project-name.git
```

2. **Navigasi ke direktori proyek**:

```
cd project-name
```

3. **Instal dependensi** (jika diperlukan):
   - Jika menggunakan **Maven**:

```
mvn install
```

   - Jika menggunakan **Gradle**:

```
gradle build
```

4. **Jalankan aplikasi**:
   - Untuk Maven:

```
mvn exec:java
```

   - Untuk Gradle:

```
gradle run
```

REFERENSI

Tutorial Menggunakan Git dalam IntelliJ IDEA
Tutorial Pengelolaan Dependensi dengan Maven
Tutorial Membuat File README yang Baik
Tutorial Menggunakan JavaDoc dalam IntelliJ IDEA

CODELAB

CODELAB 1

Buat repositori Git baru di platform penyimpanan kode seperti GitHub. Inisialisasi proyek Java sederhana dalam repositori tersebut. Buat beberapa perubahan dalam kode, lakukan commit, dan dorong perubahan ke repositori.

Contoh Ouput :

```
Cloning into 'JavaVersionControlPractice'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.

[master c1b32f3] Added a simple Java program
 1 file changed, 2 insertions(+)
 create mode 100644 Main.java
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 247 bytes | 247.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0)
To <URL_repositori>
   c1b32f3..c30536c  master -> master
```

CODELAB 2

Buatlah file README dalam proyek "DocumentationPractice". Tulis panduan penggunaan proyek dan informasi tentang proyek tersebut.

TUGAS

TUGAS 1

Buatlah sebuah aplikasi sederhana nota pemesanan makanan di sebuah restoran, model dan fitur setiap praktikan bebas (jika antar praktikan ada yang sama akan dilakukan pengurangan nilai). Pada pembuatan aplikasi ini, implementasikan beberapa atribut dan method, dan selama penulisan kode, manfaatkan fitur autocomplete untuk mengisi kode lebih cepat. Buatlah Custom Live Template pada blok kode rumus perhitungan, cobalah menggunakan custom live template yang telah dibuat.

TUGAS 2

Gunakan program yang sudah kalian buat di modul 2 (sesuai dengan spreadsheet dan sudah di refactoring). Buatlah dokumentasi dan file README.md untuk kelas dan method pada program menggunakan JavaDoc.

TUGAS 3

Buatlah repositori Git baru untuk program pada Tugas 2 yang telah kamu buat pada tugas sebelumnya. Implementasikan beberapa fitur atau fungsionalitas tambahan pada proyek tersebut. Gunakan branch dalam Git untuk mengelola pengembangan fitur, (tunjukkan caranya ke asisten masing-masing secara **Live demo**).

KRITERIA & DETAIL PENILAIAN

A. *KRITERIA PENILAIAN UMUM*

KRITERIA PENILAIAN	POIN (TOTAL 100%)
CODELAB 1	**Total 10%**
Dapat membuat Repository	3%
Dapat melakukan commit project	3%
Dapat membuat perubahan dan push	4%
CODELAB 2	**Total 10%**
Dapat membuat file README	3%
Dapat membuat penjelasan program	3%
Output File README sesuai	4%
TUGAS 1	**Total 25%**
Implementasi Autocomplete	10%
Implementasi Custom Live Templates	10%
Pemahaman Materi	5%
TUGAS 2	**Total 25%**
Implementasi JavaDoc	10%
Implementasi README	10%
Pemahaman Materi	5%
TUGAS 3	**Total 30%**
Membuat repositori	5%
Implementasi Fitur Git	20%
Pemahaman Materi	5%