VERSION 1.0

JULY 29, 2023

PEMROGRAMAN LANJUT

MODUL 3 – MODERN PROGRAMMING ENVIRONMENT AND DOCUMENTATION STYLE

TIM PENYUSUN:

- WILDAN SUHARSO, S.KOM., M.KOM
- LARYNT SAWFA KENANGA
- AHYA NIKA SALSABILA

LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN LANJUT

PERSIAPAN MATERI

Praktikan harus memahami konsep dasar tentang lingkungan pemrograman modern dan gaya dokumentasi dalam bahasa pemrograman Java. Materi yang harus dipelajari mencakup:

Modern Programming Environment:

1. Konsep IDE (Integrated Development Environment) dan fungsinya.
2. Pengenalan Git untuk pengendalian versi.
3. Penggunaan Dependency Management (Contoh: Maven atau Gradle).
4. Memahami Continuous Integration/Continuous Deployment (CI/CD) dalam pengembangan perangkat lunak.

Documentation Style:

1. Pentingnya dokumentasi dalam pengembangan perangkat lunak.
2. Jenis-jenis dokumentasi (Kode sumber, Javadoc, README, dll.).
3. Penggunaan komentar dalam kode (JavaDoc, komentar satu baris, komentar multi-baris).
4. Struktur dan konten dari file README untuk proyek.

TUJUAN

1. Mahasiswa mampu memahami dan mengoperasikan Integrated Development Environment (IDE) untuk pengembangan perangkat lunak.
2. Mahasiswa mampu menggunakan alat pengelolaan dependensi seperti Maven atau Gradle untuk mengelola komponen dan pustaka eksternal dalam proyek.
3. Mahasiswa mampu mengimplementasikan pengendalian versi menggunakan Git dan memahami konsep dasar Continuous Integration/Continuous Deployment (CI/CD).
4. Mahasiswa mampu menyusun dokumentasi yang efektif dengan mengenali jenis-jenis dokumentasi, penggunaan komentar dalam kode, dan struktur yang sesuai dalam file README.

5. Mahasiswa siap bekerja dalam lingkungan pemrograman modern, memanfaatkan alat yang ada, dan menghasilkan perangkat lunak berkualitas tinggi dengan dokumentasi yang baik.

TARGET MODUL

Penguasaan materi ini akan diukur melalui modul akhir yang mengharuskan praktikan untuk membuat proyek Java lengkap dengan dependensi, menggunakan Git untuk pengendalian versi, dan menghasilkan dokumentasi yang sesuai.

PERSIAPAN SOFTWARE/APLIKASI

1. Java Development Kit.
2. Text Editor / IDE (**Preferably** Intellij IDEA, Visual Studio Code, Netbeans, etc).
3. Git (untuk kontrol versi)
4. Maven atau Gradle (Dependency Management)

TEORI

A. Modern Programming Environment

Dalam pengembangan perangkat lunak modern, IDE seperti IntelliJ IDEA atau Eclipse memfasilitasi pengembang dengan fitur-fitur seperti debugging, autocomplete, dan integrasi Git. Praktikan akan mempelajari cara membuat proyek, menambahkan dependensi menggunakan Maven atau Gradle, serta mengintegrasikan perubahan ke repositori Git. Praktikan juga akan memahami konsep CI/CD untuk memastikan perubahan kode diuji dan diterapkan dengan benar.

1. Konsep IDE (Integrated Development Environment) dan Fungsinya

IDE adalah lingkungan pemrograman yang menyatukan berbagai alat untuk memudahkan pengembangan perangkat lunak. Ini mencakup penyunting kode, pengelola proyek, penyusun kode, debugger, dan lain-lain dalam satu tampilan yang terintegrasi. IDE membantu meningkatkan produktivitas pengembang dengan menyediakan alat yang diperlukan dalam satu tempat.

2. Fitur-Fitur IDE

Setiap IDE memiliki masing-masing fitur yang dimiliki, berikut merupakan fitur yang terdapat dalam IDE Intellij:

- **Debugging**

Debugging adalah proses mengidentifikasi dan memperbaiki bug atau kesalahan dalam kode. IntelliJ IDEA menyediakan alat bantu debugging yang kuat untuk membantu kamu melacak perubahan nilai variabel, melihat jejak pemanggilan (call stack), dan mengidentifikasi masalah dalam kode.

Cara Menggunakan Debugging:

1) Letakkan breakpoint (titik henti) pada baris kode yang ingin kamu periksa.
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.

Contoh:

Misalkan kamu memiliki kode berikut:

```
public class DebugExample {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int sum = x + y;
        System.out.println("Sum: " + sum);
    }
}
```

Kamu dapat menempatkan breakpoint pada baris int sum = x + y;, menjalankan program dalam mode debug, dan mengamati nilai variabel serta langkah eksekusi.

- **Autocomplete**

Autocomplete (pengisian otomatis) adalah fitur yang memprediksi kata atau kode yang ingin Anda tulis dan menawarkan opsi untuk melengkapi kata tersebut. Ini membantu meningkatkan produktivitas dan mengurangi kesalahan pengetikan.

Cara Menggunakan Autocomplete:

1) Mulai mengetik kode atau kata kunci.
2) IntelliJ IDEA akan menampilkan pilihan autocomplete di bawah input Anda.
3) Pilih pilihan yang diinginkan dengan menggunakan tombol panah atau klik mouse.

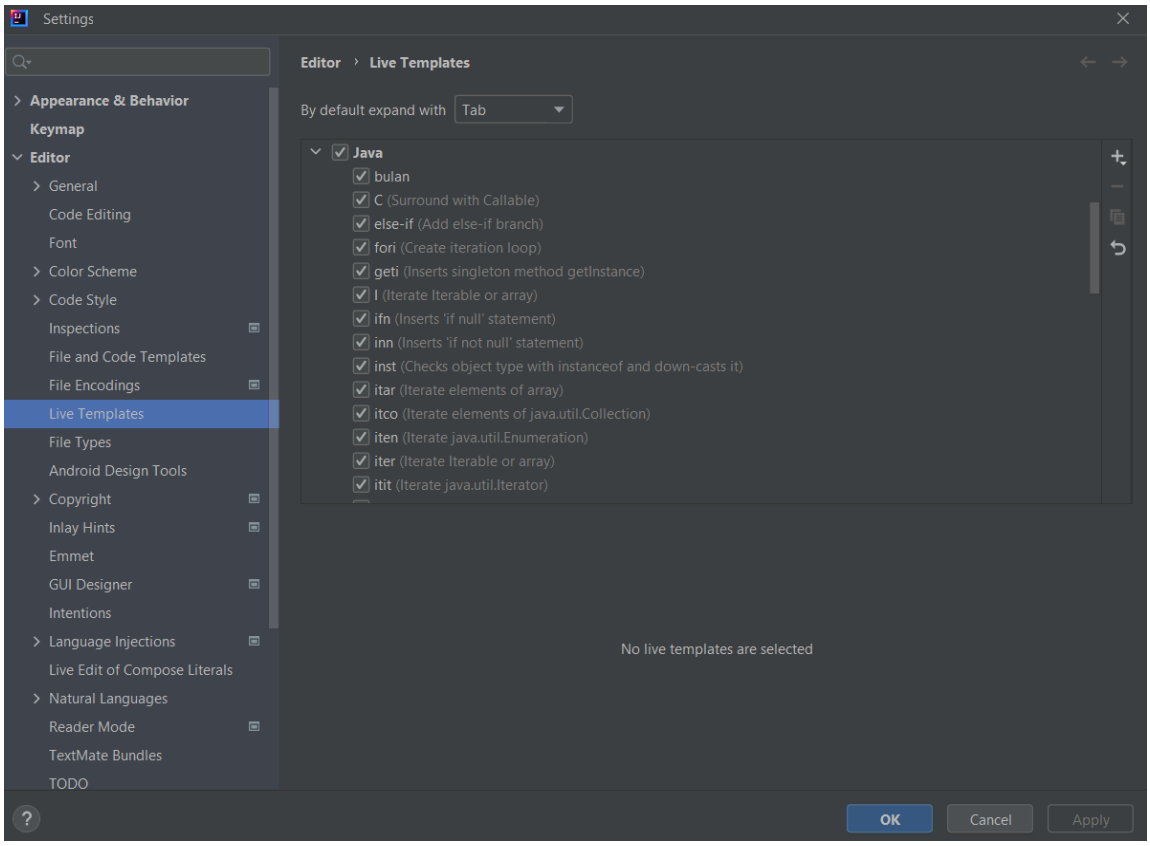
Contoh:

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

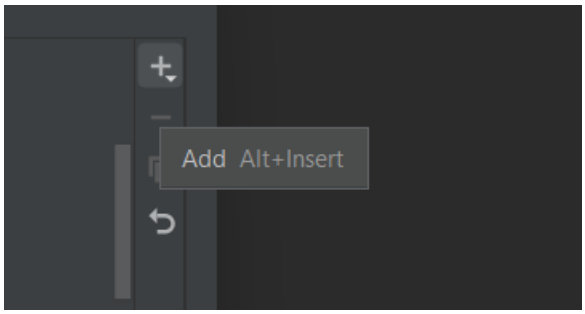
psvm	Membuat metode public static void main(String[] args) dengan mengetik "psvm" diikuti dengan tombol Tab.
fori	Membuat perulangan for dengan mengisi variabel dan batasan indeks dengan mengetik "fori" diikuti dengan tombol Tab.
iter	Membuat perulangan for-each untuk mengiterasi elemen dalam koleksi 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:

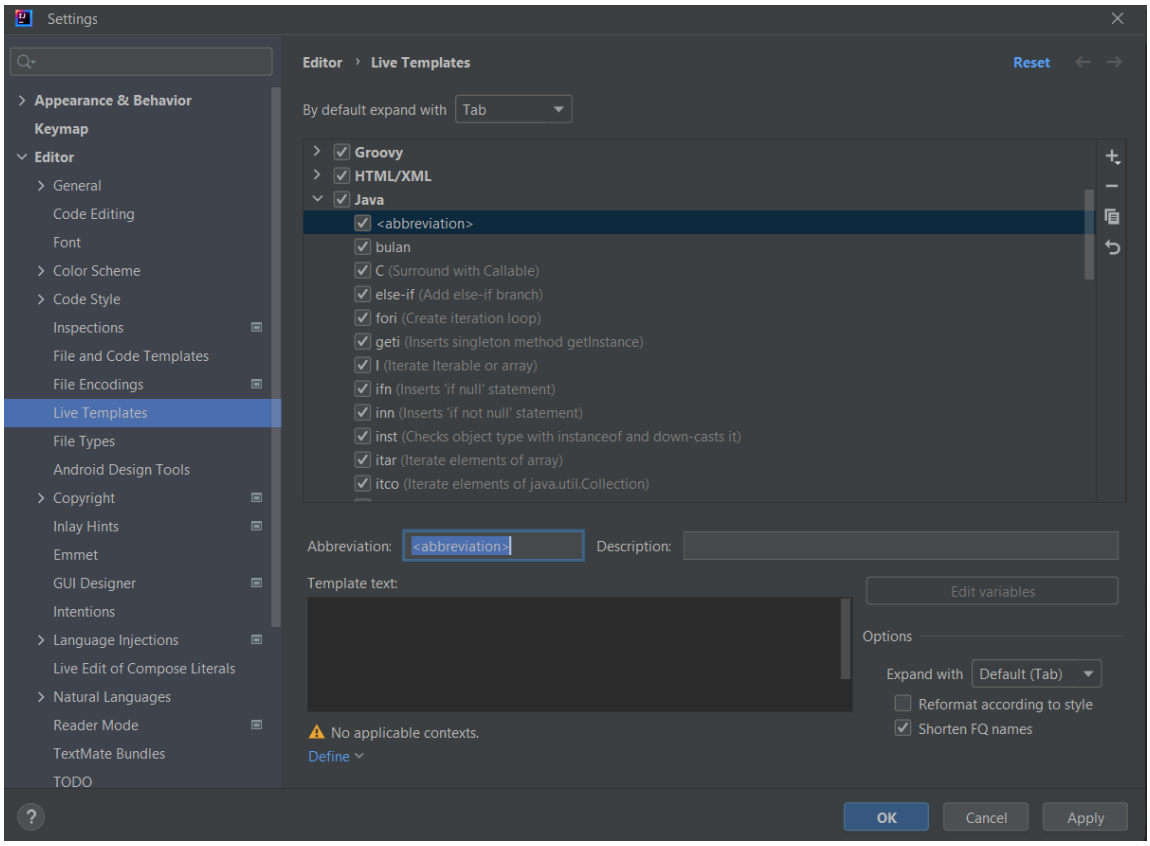
Ctrl+Alt+S atau dari File > Settings. Setelah itu akan muncul pop up berikut. Pilih Editor dan klik Live Templates.

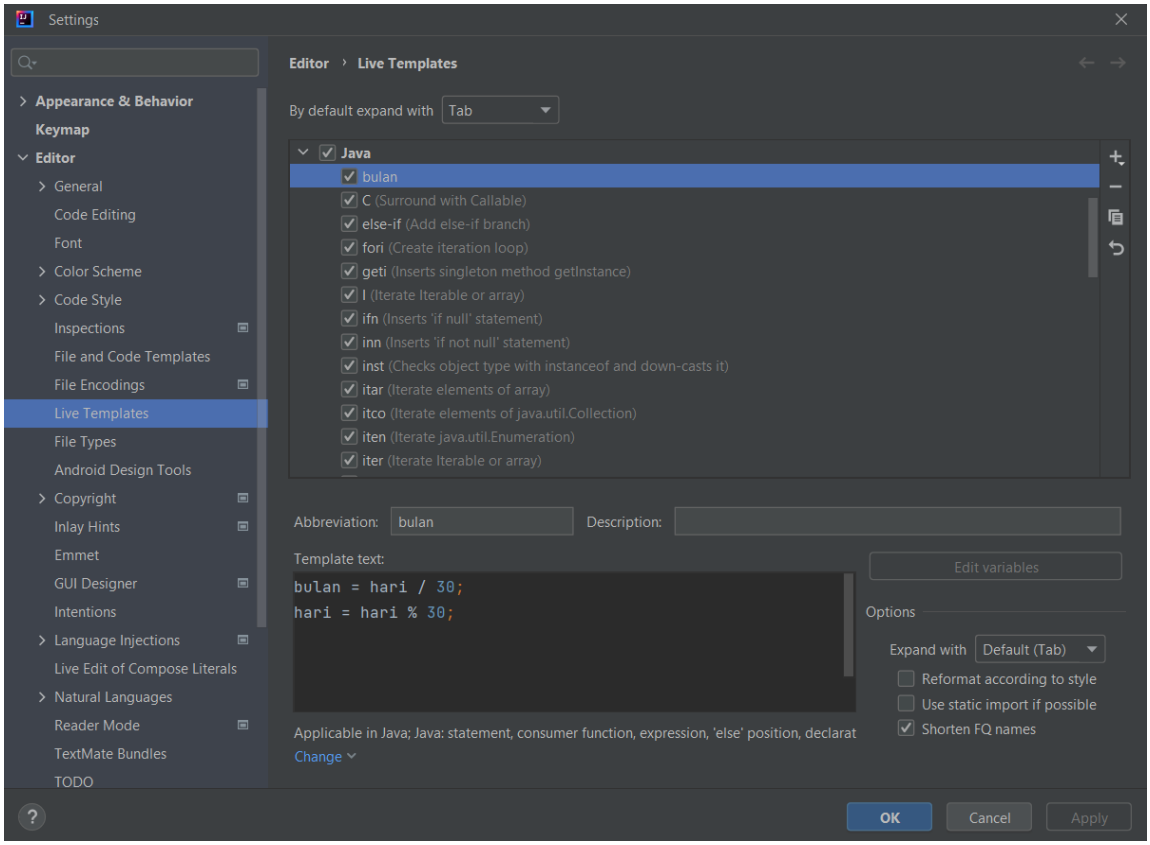


Kamu akan diarahkan ke bagian live templates, klik Java dan klik tanda panah untuk menambahkan template kalian sendiri.



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


Settings
Editor › Live Templates
Reset
By default expand with Tab
Groovy
HTML/XML
Java
<abbreviation>
bulan
C (Surround with Callable)
else-if (Add else-if branch)
fori (Create iteration loop)
geti (Inserts singleton method getInstance)
I (Iterate Iterable or array)
ifn (Inserts 'if null' statement)
inn (Inserts 'if not null' statement)
inst (Checks object type with instanceof and down-casts it)
itar (Iterate elements of array)
itco (Iterate elements of java.util.Collection)
Abbreviation: <abbreviation>
Description:
Template text:
Edit variables
Options
Expand with Default (Tab)
Reformat according to style
Shorten FQ names
No applicable contexts.
Define
OK
Cancel
Apply


Settings
Editor › Live Templates
By default expand with Tab
Java
bulan
C (Surround with Callable)
else-if (Add else-if branch)
fori (Create iteration loop)
geti (Inserts singleton method getInstance)
I (Iterate Iterable or array)
ifn (Inserts 'if null' statement)
inn (Inserts 'if not null' statement)
inst (Checks object type with instanceof and down-casts it)
itar (Iterate elements of array)
itco (Iterate elements of java.util.Collection)
iten (Iterate java.util.Enumeration)
iter (Iterate Iterable or array)
Abbreviation: bulan
Description:
Template text:
bulan = hari / 30;
hari = hari % 30;
Edit variables
Options
Expand with Default (Tab)
Reformat according to style
Use static import if possible
Shorten FQ names
Applicable in Java; Java: statement, consumer function, expression, 'else' position, declarat
Change
OK
Cancel
Apply

- **Integrasi Git**

Integrasi Git dalam IntelliJ IDEA memungkinkan Anda untuk mengelola kode sumber melalui antarmuka pengguna IDE tanpa perlu beralih ke terminal atau aplikasi lain. Anda dapat melakukan 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 kanan, 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 (Contoh: Maven atau Gradle)**

Dependency Management adalah praktik untuk mengelola dependensi eksternal yang diperlukan oleh proyek Anda. Alat seperti Maven atau Gradle membantu mengotomatisasi proses mengunduh, mengelola, dan mengintegrasikan dependensi ke dalam proyek Anda. Ini memungkinkan Anda untuk fokus pada pengembangan fitur dan fungsionalitas unik Anda tanpa perlu khawatir tentang mengelola dependensi secara manual.

Cara Menggunakan Dependency Management:

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 pendekatan dalam pengembangan perangkat lunak yang melibatkan otomatisasi pengujian, integrasi kode, dan

pengiriman perubahan ke lingkungan produksi. CI/CD membantu menjaga kualitas kode, mengurangi risiko kesalahan, dan memungkinkan pengembangan yang lebih cepat dan aman.

Cara Memahami CI/CD:

1) Memahami CI: CI melibatkan menggabungkan kode dari berbagai anggota tim secara teratur, diikuti oleh otomatisasi pengujian unit dan integrasi untuk mendeteksi masalah lebih awal.
2) Memahami CD: CD melibatkan pengiriman perubahan kode yang telah lulus uji ke lingkungan produksi secara otomatis atau dengan intervensi minimal.

Penerapan di IntelliJ IDEA:

1) Menggunakan Jenkins atau alat CI/CD lainnya, Anda dapat mengonfigurasi otomatisasi pengujian dan pengiriman perubahan kode dari repositori ke lingkungan produksi.
2) IntelliJ IDEA memiliki integrasi dengan alat CI/CD seperti Jenkins yang memungkinkan Anda mengelola proses CI/CD langsung dari dalam IDE.

Untuk lebih lengkapnya, teman-teman bisa mengunjungi situs dibawah ini:

Tutorial Menggunakan Git dalam IntelliJ IDEA

Tutorial Pengelolaan Dependensi dengan Maven

Tutorial Menggunakan Continuous Integration dengan Jenkins

B. Documentation Style

Dokumentasi adalah komponen kunci dalam pengembangan perangkat lunak. Dokumentasi yang baik membantu memahami tujuan proyek, fungsionalitas, dan cara penggunaan, yang semuanya penting bagi pengembang dan pengguna.

1. Jenis-Jenis Dokumentasi

- **Kode Sumber**

Kode sumber adalah komentar yang menjelaskan bagaimana dan mengapa suatu kode diimplementasikan.

Contoh:

```
2 usages
class Graph {
    //Ini adalah komentar
    //Deklar var
    6 usages
    private Map<Integer, List<Integer>> graph;

    //Fungsi graph untuk ....
    1 usage
    public Graph() {
        graph = new HashMap<>();
    }
```

- **JavaDoc**

Komentar yang memungkinkan menghasilkan dokumentasi otomatis. Menggunakan tag seperti @param, @return, dll.

Contoh:

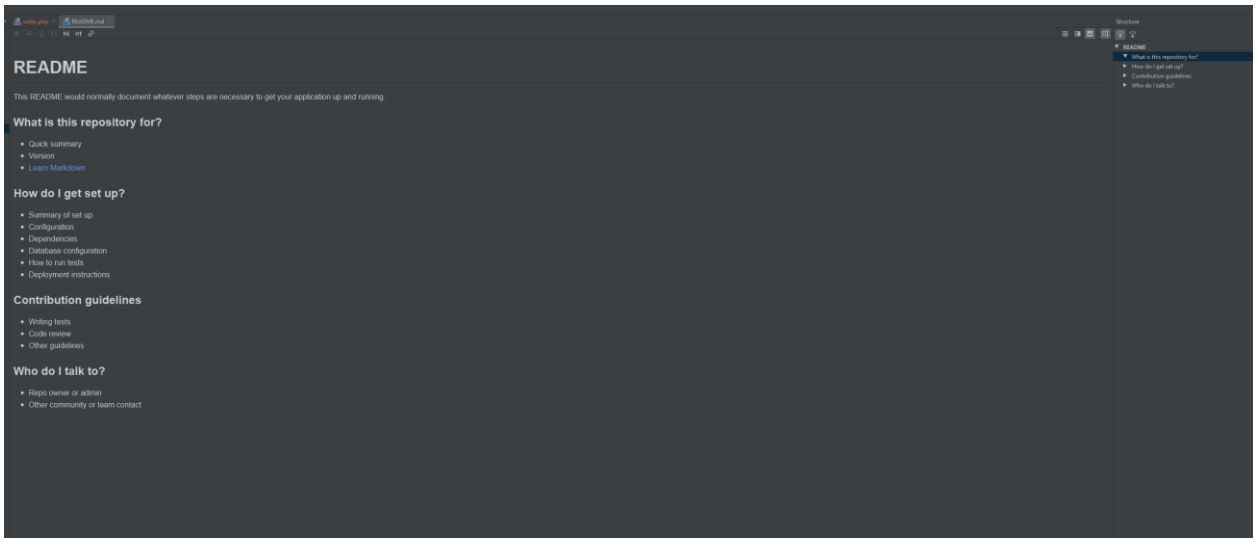
Mahasiswa.java

```
/**
 * Kelas yang merepresentasikan Mahasiswa.
 */
public class Mahasiswa {
    2 usages
    private String nama;
    2 usages
    private int nim;

    /**
     * Constructor untuk objek kelas Mahasiswa.
     *
     * @param nama Nama mahasiswa.
     * @param nim  NIM mahasiswa.
     */
    public Mahasiswa(String nama, int nim) {
        this.nama = nama;
        this.nim = nim;
    }

    /**
     * Mendapatkan nama mahasiswa.
     *
     * @return Nama mahasiswa.
     */
    public String getNama() {
        return nama;
    }
```

- **README**

README biasanya merupakan dokumen panduan yang berisi deskripsi proyek, instalasi, penggunaan, dan kontribusi.

Contoh:



Untuk lebih lengkap bisa dilihat disitus:

Tutorial Menggunakan JavaDoc dalam IntelliJ IDEA

Tutorial Membuat File README yang Baik

LATIHAN

LATIHAN 1: Menggunakan Git untuk Pengendalian Versi
Buat repositori Git baru di platform penyimpanan kode seperti GitHub atau GitLab. Inisialisasi proyek Java sederhana dalam repositori tersebut. Buat beberapa perubahan dalam kode, lakukan commit, dan dorong perubahan ke repositori.

```
# Ubah direktori kerja ke tempat Anda ingin menyimpan proyek
cd /path/to/your/workspace
# Clone repositori ke komputer Anda
git clone <URL_repositori>
# Masuk ke direktori proyek yang telah dicloning
cd JavaVersionControlPractice

# Buat file Java sederhana (Contoh: Main.java)
```

```
# Isi file dengan kode sederhana, misal:
# public class Main {
#     public static void main(String[] args) {
#         System.out.println("Hello, Git!");
#     }
# }

# Tambahkan perubahan ke indeks Git
git add Main.java

# Buat commit dengan pesan
git commit -m "Added a simple Java program"

# Dorong perubahan ke repositori jarak jauh
git push origin master
```

Output:

```
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
```

LATIHAN 2: Membuat dan Mengelola Dependensi dengan Maven

Soal: Buat proyek Maven baru dengan nama "DependencyPractice". Tambahkan dependensi dari Apache Commons Lang versi 3.12.0 ke proyek. Buat program sederhana yang menggunakan fungsi dari dependensi tersebut.

```
# Ubah direktori kerja ke tempat Anda ingin menyimpan proyek
cd /path/to/your/workspace

# Buat proyek Maven baru
mvn archetype:generate -DgroupId=com.example -DartifactId=DependencyPractice -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

# Masuk ke direktori proyek
cd DependencyPractice

# Buka file pom.xml dan tambahkan dependensi Apache Commons Lang
# <dependencies>
#     <dependency>
#         <groupId>org.apache.commons</groupId>
#         <artifactId>commons-lang3</artifactId>
#         <version>3.12.0</version>
#     </dependency>
# </dependencies>

# Buat program sederhana yang menggunakan fungsi dari dependensi
# Buat file Java baru (Contoh: App.java)
# Isi file dengan kode sederhana, misal:
# import org.apache.commons.lang3.StringUtils;
#
# public class App {
#     public static void main(String[] args) {
#         String text = "Hello, Maven!";
#         String reversedText = StringUtils.reverse(text);
#         System.out.println(reversedText);
#     }
# }

# Kompilasi dan jalankan program
mvn compile
mvn exec:java -Dexec.mainClass="com.example.App"
```

Output:

```
[INFO] Scanning for projects...
[INFO]
[INFO] --------------< com.example:DependencyPractice >--------------
[INFO] Building DependencyPractice 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ Depenc
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /path/to/your/workspace/Dependenc
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ Dependenc
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /path/to/your/workspace/DependencyPractice
[INFO]
[INFO] --- exec-maven-plugin:3.0.0:java (default-cli) @ DependencyPractice -
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  2.356 s
[INFO] Finished at: 2023-08-23T12:34:56+00:00
[INFO] ------------------------------------------------------------------------
Hello, Maven!
```

LATIHAN 3: Menyusun Dokumentasi dalam File README

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

```
# Ubah direktori kerja ke tempat Anda ingin menyimpan proyek
cd /path/to/your/workspace
```

```
# Buat direktori baru untuk proyek
mkdir DocumentationPractice

# Masuk ke direktori proyek
cd DocumentationPractice

# Buat file README.md dan tulis konten
# Contoh:
# # DocumentationPractice
# This is a simple project to practice creating documentation in a README file.
#
# ## Usage
# 1. Clone the repository to your local machine.
# 2. Open the project in your favorite IDE.
# 3. Run the program to see the output.
#
# ## Dependencies
# - None
#
# ## License
# This project is licensed under the MIT License - see the [LICENSE](LICENSE) file for details.

# Simpan dan tutup file README.md
```

Output :

Buka file README.md di editor teks atau tampilan GitHub untuk melihat konten yang telah ditulis.

TUGAS

TUGAS 1: Implementasi Autocomplete dan Custom Live Template

Implementasi Autocomplete dan Custom Live Template

Instruksi: Buat sebuah program sederhana menggunakan Java dalam proyek IntelliJ IDEA. Implementasikan fitur autocomplete dalam kode dan buat juga sebuah custom live template untuk memudahkan penulisan kode tertentu.

Langkah-langkah:

1. Buat Proyek:
   - Buka IntelliJ IDEA dan buat proyek baru dengan nama "AutocompleteAndTemplates".
   - Pilih "Java" sebagai tipe proyek.
2. Implementasi Autocomplete:
   - Buka file Main.java yang ada dalam proyek.
   - Implementasikan sebuah kelas sederhana yang memiliki beberapa atribut dan metode.
   - Selama penulisan kode, manfaatkan fitur autocomplete untuk mengisi kode lebih cepat.
3. Custom Live Template:
   - Buka pengaturan live template di IntelliJ IDEA.
   - Tambahkan sebuah custom live template dengan nama sesuai nama fungsi.
   - Konfigurasikan template tersebut untuk menghasilkan blok kode rumus perhitungan.
4. Menggunakan Custom Live Template:
   - Buka file Main.java lagi.
   - Cobalah menggunakan custom live template yang telah di buat untuk membuat kalkulator sederhana.

TUGAS 2: Java Class dengan JavaDoc

Buatlah sebuah kelas Java yang mengimplementasikan fitur sederhana (setiap orang harus menggunakan tema yang berbeda), misalnya penghitungan luas dan keliling lingkaran. Gunakan JavaDoc untuk mendokumentasikan kelas dan metode yang kamu buat.

TUGAS 3: Implementasi Fitur dengan Git

Buatlah repositori Git baru untuk proyek Java 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**).

TUGAS 4: Dokumentasi dengan README.md

Buatlah file README.md untuk proyek Java kamu. Dokumentasikan cara menjalankan proyek, deskripsi singkat tentang proyek, dan informasi lain yang bermanfaat bagi pengguna proyek.

REFERENSI

Tutorial Menggunakan Git dalam IntelliJ IDEA

Tutorial Pengelolaan Dependensi dengan Maven

Tutorial Menggunakan Continuous Integration dengan Jenkins

Tutorial Menggunakan JavaDoc dalam IntelliJ IDEA

Tutorial Membuat File README yang Baik

KRITERIA DAN PENILAIAN

Kriteria Penilaian		Nilai
Latihan 1		12.5
Dapat membuat repository baru	2.5	
Dapat melakukan commit project	5	
Dapat membuat perubahan dan push perubahan	5	
Latihan 2		12.5
Dapat membuat project Maven	2.5	
Menambahkan depedency	5	
Dapat mengcompile dan menjalankan program	5	
Latihan 3		12.5
Dapat membuat file README	2.5	
Dapat membuat penjelasan tentang program	5	
Dapat memperlihatkan hasil file README	5	
Tugas 1		10

Implementasi Autocompile	5	
Custom Live Templates	5	
Tugas 2		7.5
Menggunakan JavaDoc dengan lengkap	5	
Memberikan komentar	2.5	
Tugas 3		5
Mampu memperlihatkan cara penggunaan fitur	5	
Tugas 4		10
Membuat file README	5	
Isi file lengkap (how to operate/use, description, important informations)	5	
Pemahaman		30
Total		100