

VERSION 1.0

AGUSTUS 7, 2024



PEMROGRAMAN LANJUT

MODUL 3 – MODERN PROGRAMMING ENVIRONMENT AND DOCUMENTATION STYLE

DISUSUN OLEH:

- WILDAN SUHARSO, S.KOM, M.KOM

- LUTHFIA SHOFA DEWI

- RAHMATUN NIKMAH

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.

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**

IDE (Integrated Development Environment) adalah sebuah software aplikasi biasanya berbasis GUI, dan digunakan untuk menulis dan running baris kode pemrograman. IDE seperti IntelliJ IDEA atau Eclipse yang kita akan gunakan pada mata kuliah pemrograman lanjut ini sudah menyediakan 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

Selain untuk menulis dan running kode pemrograman, IDE digunakan untuk meningkatkan produktivitas developer dengan menggabungkan kemampuan seperti pengeditan, pembangunan, pengujian, dan pengemasan perangkat lunak dalam aplikasi yang mudah dilakukan. IDE juga menyediakan beberapa tools yang diperlukan oleh programmer.

2. Fitur-Fitur IDE

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

- **Debugging**

Debugging adalah proses mendeteksi dan memperbaiki kesalahan dalam suatu program. Ada berbagai jenis kesalahan yang dapat diatasi dengan debugging, beberapa diantaranya adalah kesalahan sintaksis. Kesalahan sederhana dapat segera diidentifikasi dengan melihat call stack yang membantu menentukan lokasi kesalahan. Tetapi, ada juga kesalahan yang jauh lebih rumit dan memerlukan waktu yang lebih lama untuk diidentifikasi dan diperbaiki. Disinilah debugging menjadi berguna. Debugging dilakukan dengan menghentikan sementara eksekusi dan menganalisis keadaan program, termasuk memeriksa variabel dan perubahan yang terjadi baris demi baris.

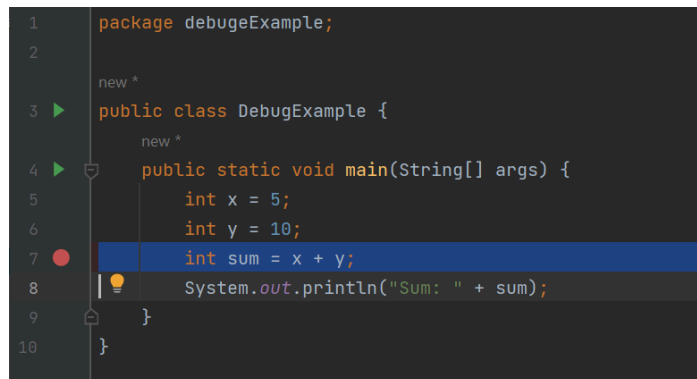
Contoh:

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);
    }
}
```

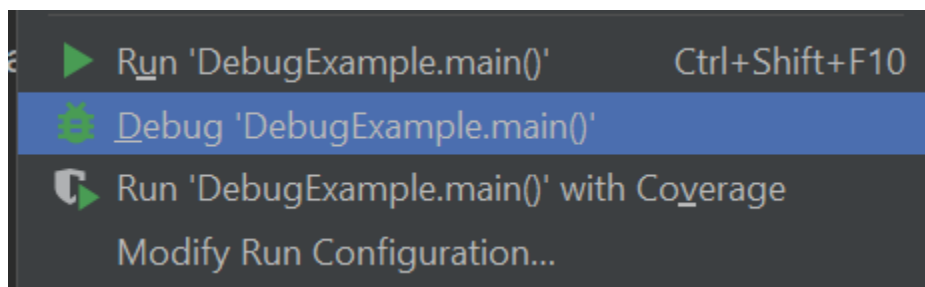
Cara Menggunakan Debugging:

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

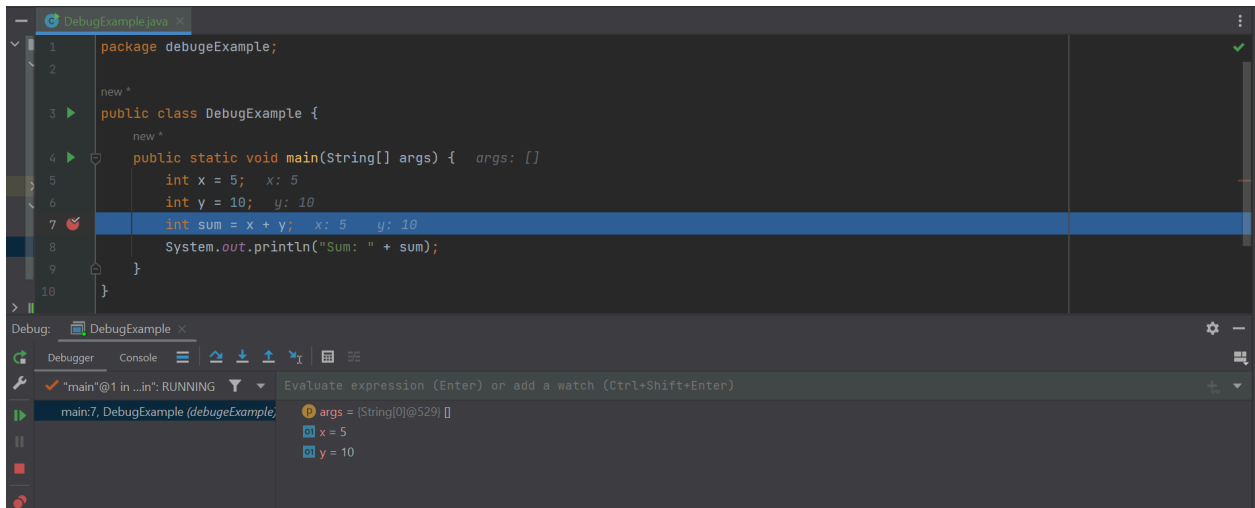


caranya, dengan klik Gutter (klik area di sebelah kiri nomor). Klik di nomor baris yang kamu pilih. Kamu akan melihat titik merah muncul, menandakan bahwa breakpoint telah ditambahkan

- 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 memudahkan penulisan kode dengan menyediakan saran otomatis untuk nama variabel, method, kelas, dan elemen kode lainnya saat kita mengetik kode. Fitur ini sangat membantu dalam meningkatkan produktivitas dan mengurangi kesalahan dalam menulis.

Contoh Cara Menggunakan Autocomplete:

- 1) Mulai mengetik kode atau kata kunci.

Misalkan disini kita akan mencoba untuk menulis `System.out.println()`, cara mudah agar kita tidak perlu menulis panjang, maka kita bisa menuliskan **sout** saja

- 2) IntelliJ IDEA akan menampilkan pilihan autocomplete di bawah input Anda.

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

sout Prints a string to System.out
soutm Prints current class and method names to System.o...
soutp Prints method parameter names and values to Syste...
soutv Prints a value to System.out

Press Ctrl+, to choose the selected (or first) suggestion and insert a dot afterwards Next Tip
```

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

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

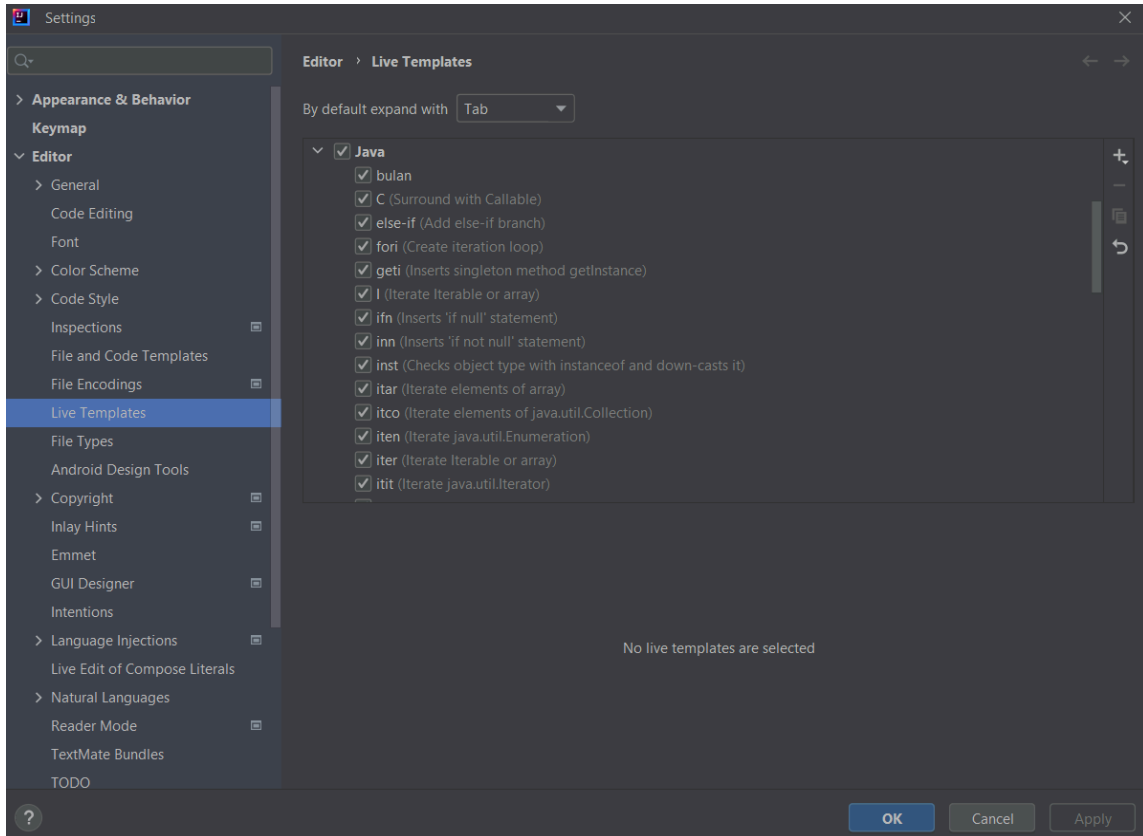
Berikut ini adalah daftar untuk keyword autocomplete

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.

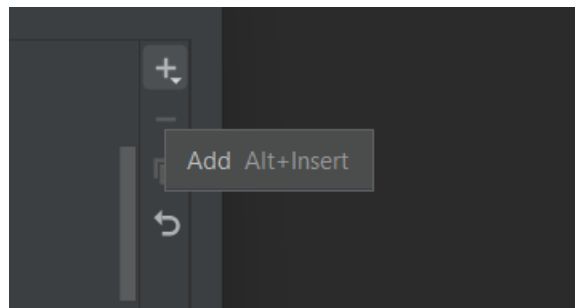
iter	Membuat perulangan for-each untuk mengiterasi elemen dalam koleksi dengan mengetik "iter" diikuti dengan tombol Tab.
ifn	Membuat pernyataan if (variable == null) untuk memeriksa apakah variabel null dengan mengetik "ifn" diikuti dengan tombol Tab.
psf	Membuat konstanta public static final dengan 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:

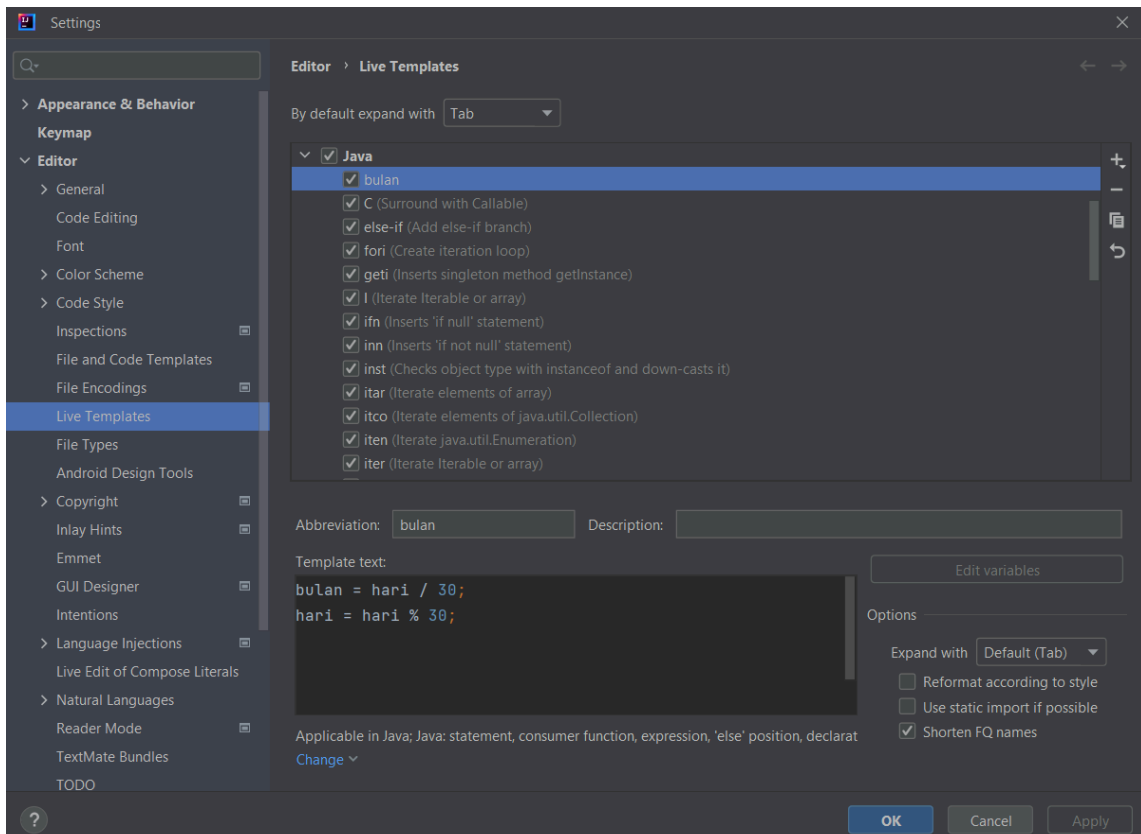
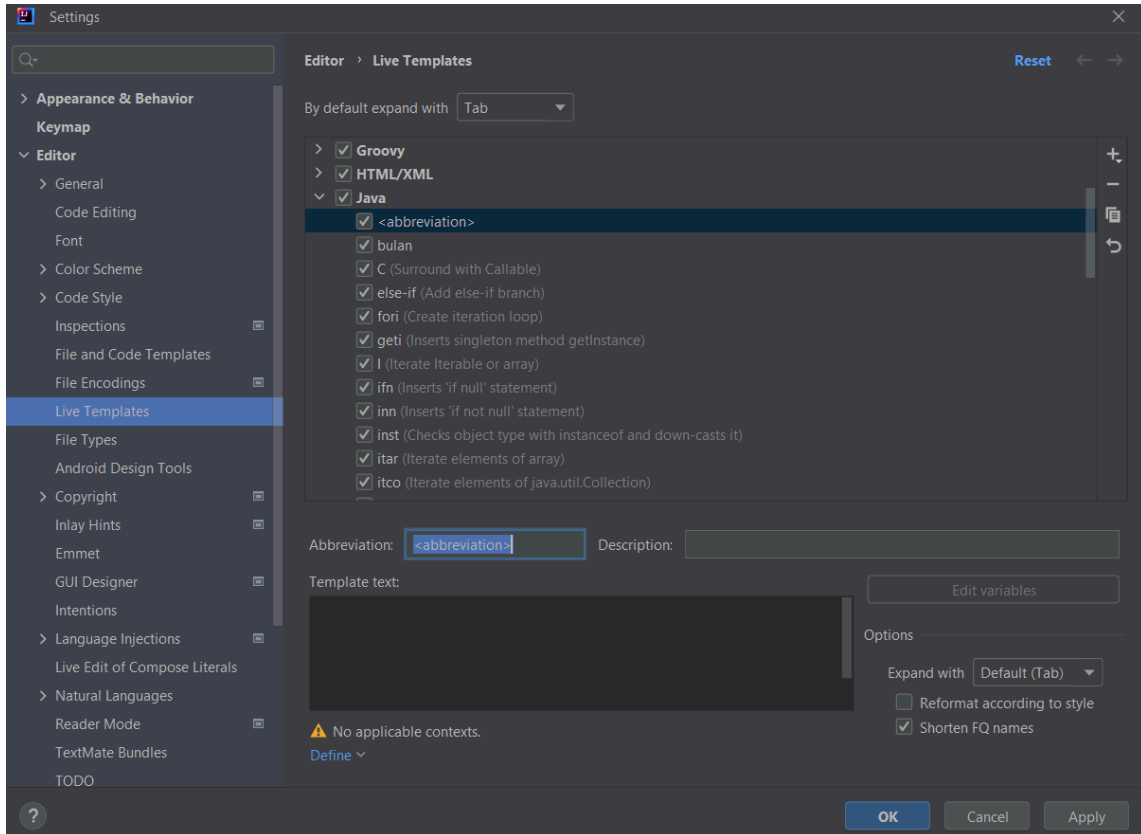
Ctrl+Alt+S atau dari File > Settings. Setelah itu akan muncul pop up berikut. Pilih Editor dan klik Live Templates.



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.

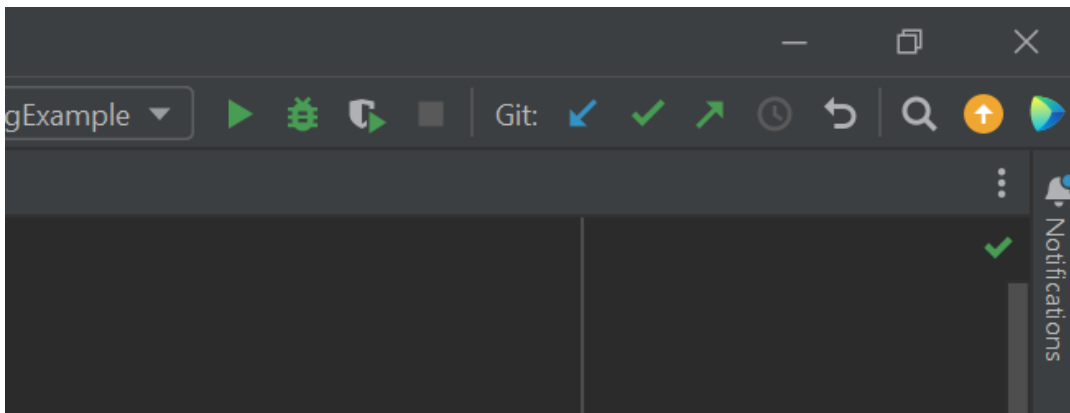


- **Integrasi Git**

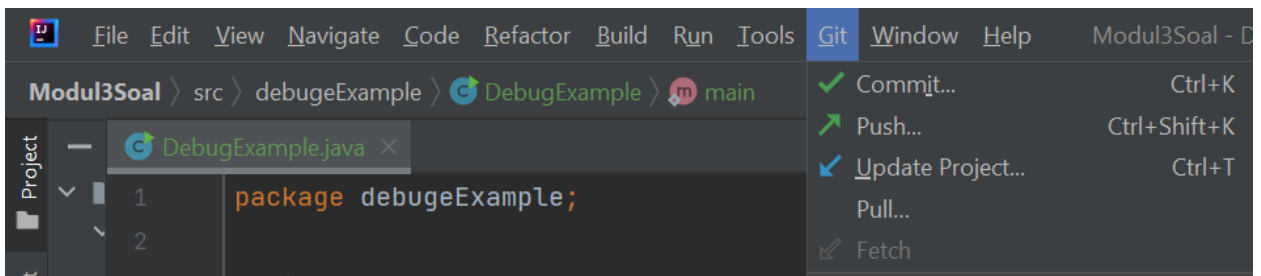
Integrasi Git dalam IntelliJ IDEA memungkinkan Anda untuk mengelola kode sumber melalui antarmuka pengguna IDE tanpa perlu beralih ke terminal atau aplikasi lain. Anda dapat melakukan operasi pengendalian versi seperti commit, push, pull, dan lainnya langsung dari dalam IntelliJ IDEA.

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.



Atau bisa pada bagian daftar menu dibagian atas IntelliJ, pilih bagian Git untuk melihat pilihan operasi pengendalian versi



- 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 mengkonfigurasi 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 proses menulis penjelasan dan informasi terkait bagian-bagian dalam sebuah proyek perangkat lunak. Tujuannya untuk mempermudah pemahaman, pemeliharaan, dan penggunaan kode oleh pengembang lain atau oleh pengembang di masa depan.

1. Jenis-jenis Dokumentasi

- **Kode Sumber**

Kode sumber adalah penjelasan singkat yang ditulis di sebelah atau atas baris kode untuk menjelaskan fungsi atau tujuan dari baris tersebut

Contoh:

```
public class DebugExample {
    new *
    public static void main(String[] args) {
        int x = 5; //nilai x
        int y = 10; //nilai y
        int sum = x + y; //hasil dari penjumlahan nilai x dan y
        System.out.println("Sum: " + sum); //menampilkan hasil penjumlahan
    }
}
```

- **JavaDoc**

JavaDoc adalah alat dokumentasi untuk bahasa pemrograman Java yang digunakan untuk menghasilkan dokumentasi terstruktur dari komentar dalam kode sumber Java. Menggunakan tag seperti @param, @return, dll. JavaDoc memungkinkan pengembang untuk mendokumentasikan kelas, metode, dan anggota lainnya dengan cara yang terstandarisasi, sehingga memudahkan pemahaman dan 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

```
public class Calculator {

    private int x;
    private int y;
    private int sum;

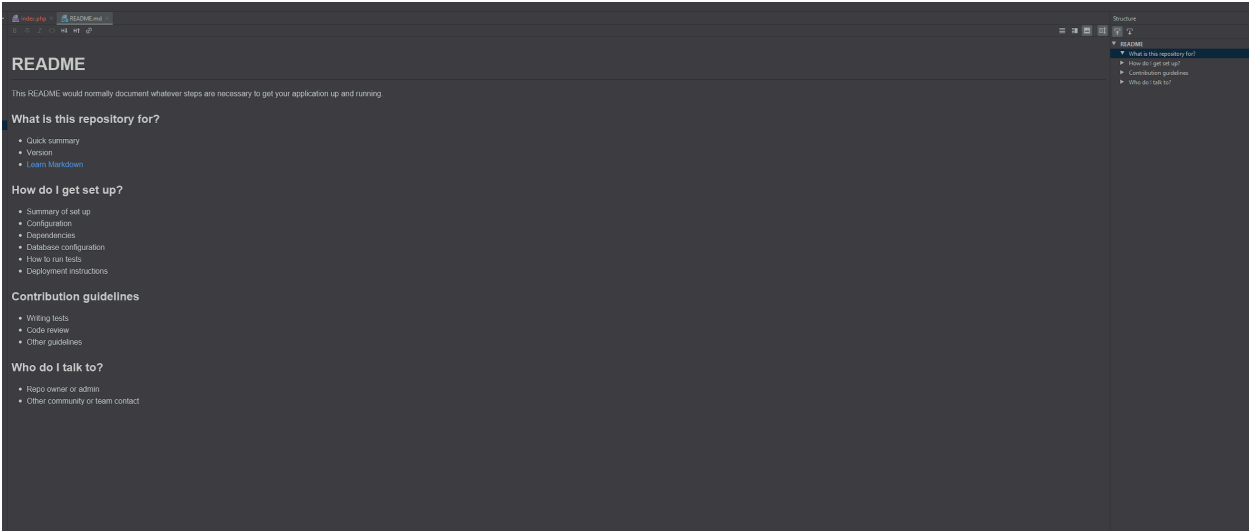
    /**
     *
     * @param x nilai x
     * @param y nilai y
     */
    new *
    public Calculator(int x, int y){
        this.x = x;
        this.y = y;
    }

    /**
     * mendapatkan nilai x
     * @return x
     */
    new *
    public int getX() {
        return x;
    }
}
```

● README

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

Contoh:



Untuk lebih lengkap bisa dilihat di situs:

[Tutorial Menggunakan JavaDoc dalam IntelliJ IDEA](#)

[Tutorial Membuat File README yang Baik](#)

CODELAB

CODELAB 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
```

CODELAB 2: MEMBUAT DAN MENGELOLA DEPENDENSI DENGAN MAVEN

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) @ DependencyPractice ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /path/to/your/workspace/DependencyPractice
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ DependencyPractice ---
[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!
```

CODELAB 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

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 proyek yang sudah kalian buat di modul 2 (sesuai dengan spreadsheet dan sudah di refactoring). Buatlah dokumentasi kelas dan method menggunakan JavaDoc pada proyek tersebut.

 Tema Pemrograman Lanjut 2024/2025

TUGAS 3

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

Buatlah file **README.md** untuk proyek yang sesuai dengan spreadsheet modul 2 dan sudah di refactoring. Dokumentasikan petunjuk untuk menjalankan proyek, berikan deskripsi singkat mengenai proyek tersebut, serta tambahkan informasi penting lainnya yang akan berguna bagi pengguna proyek ini.

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 & DETAIL PENILAIAN

Kriteria Penilaian		Nilai
Latihan 1		6
Dapat membuat repository baru	2	
Dapat melakukan commit project	2	
Dapat membuat perubahan dan push perubahan	2	
Latihan 2		6
Dapat membuat project Maven	2	
Menambahkan dependency	2	
Dapat mengcompile dan menjalankan program	2	
Latihan 3		6
Dapat membuat file README	2	
Dapat membuat penjelasan tentang program	2	
Dapat memperlihatkan hasil file README	2	
Tugas 1		12
Implementasi Autocompile	5	
Custom Live Templates	7	
Tugas 2		10
Menggunakan JavaDoc dengan lengkap	5	
Memberikan komentar	5	
Tugas 3		10
Mampu memperlihatkan cara penggunaan fitur	10	
Tugas 4		20
Membuat file README	5	
Isi file lengkap (how to operate/use, description, important informations)	15	
Pemahaman		30
Total		100