code-style-guide

1. Setup

If this is your first time setting up the code style configuration, kindly follow the steps below.

1.1. Useful options

The following options are recommended to enable automatic code formatting, rearranging, and import optimization — saving time and improving the development experience:

1. **(Recommended)**
   Go to: Use the Reformat File Dialog (Ctrl + Alt + Shift + L)
   Enable: Reformat Code, Optimize imports, Rearrange Code, Code Cleanup
2. *(Optional)*
   Settings > Appearance & Behavior > System Settings > Autosave
   Enable: Save files if the IDE is idle for 60 seconds
3. *(Optional)*
   Settings > Editor > General > Soft Wraps
   Enable: Soft Wraps these files (It is helpful if you're using `Markdown`)
4. *(Optional)*
   Settings > Languages & Frameworks > Markdown
   Enable: Markdown Extensions (`PlantUML`) (If you're using `PlantUML`)
5. *(Optional)*
   Install the `SonarQube IDE` plugin to detect code smells, security issues, maintainability problems, and potential bugs

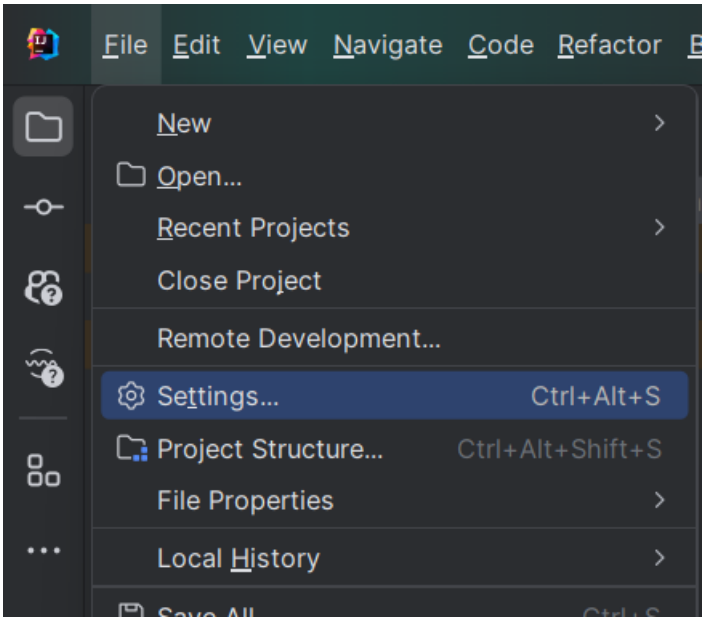
1.2. Code Style Setup

Two configuration files are provided:

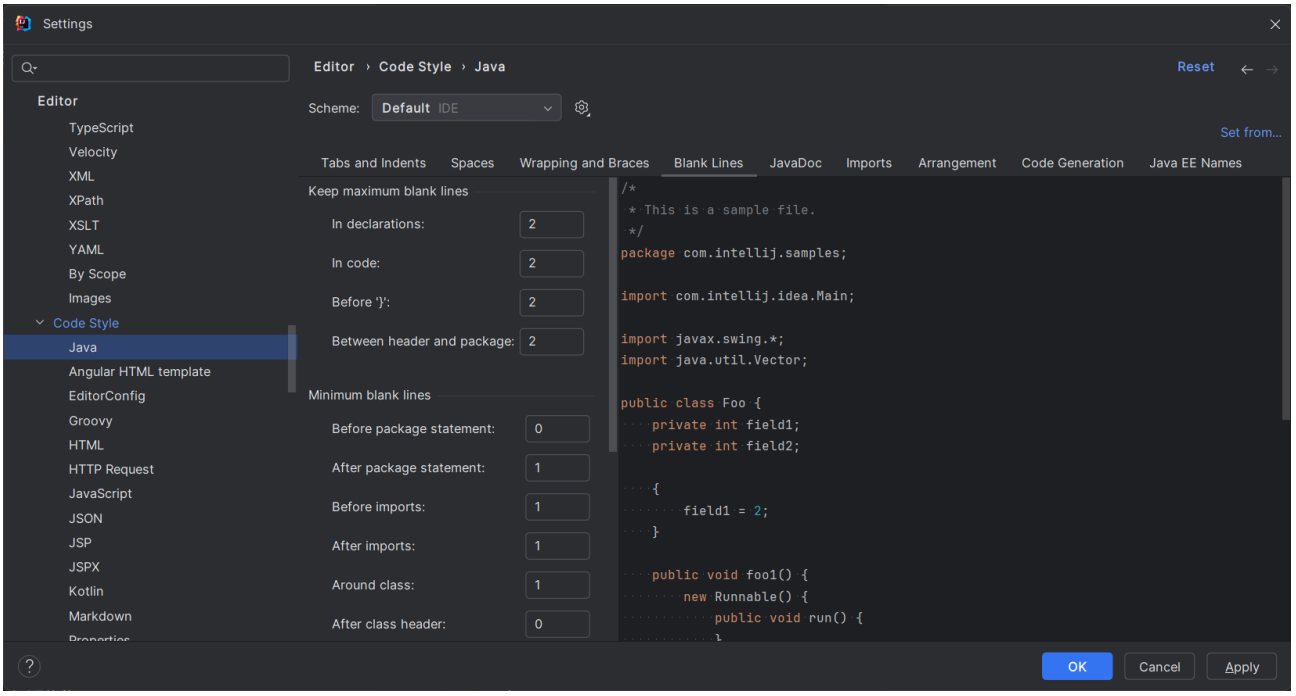
File	Usage
`iFast-Illuminator-CodeStyle.xml`	Defines formatting and arrangement rules (indentation, alignment, wrapping, etc.)
`iFast-Illuminator-InspectionProfile.xml`	Defines inspections for code smells, potential bugs, maintainability, and security issues

1.2.1. `iFastIlluminatorStyle.xml`

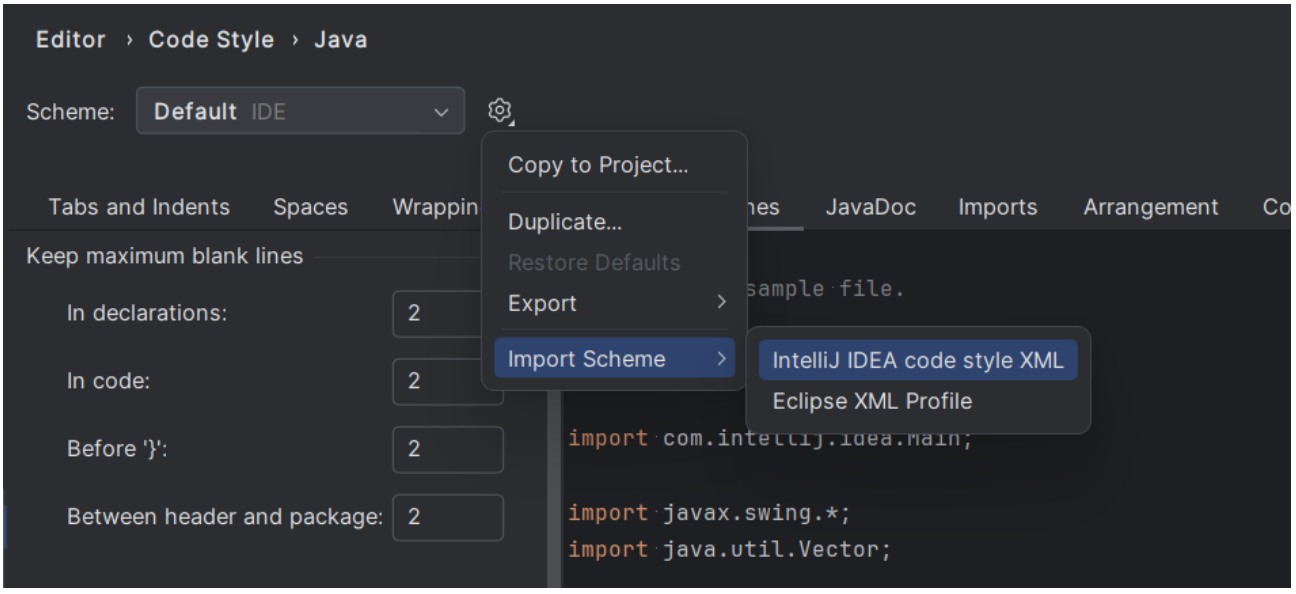
1. Open **File > Settings**



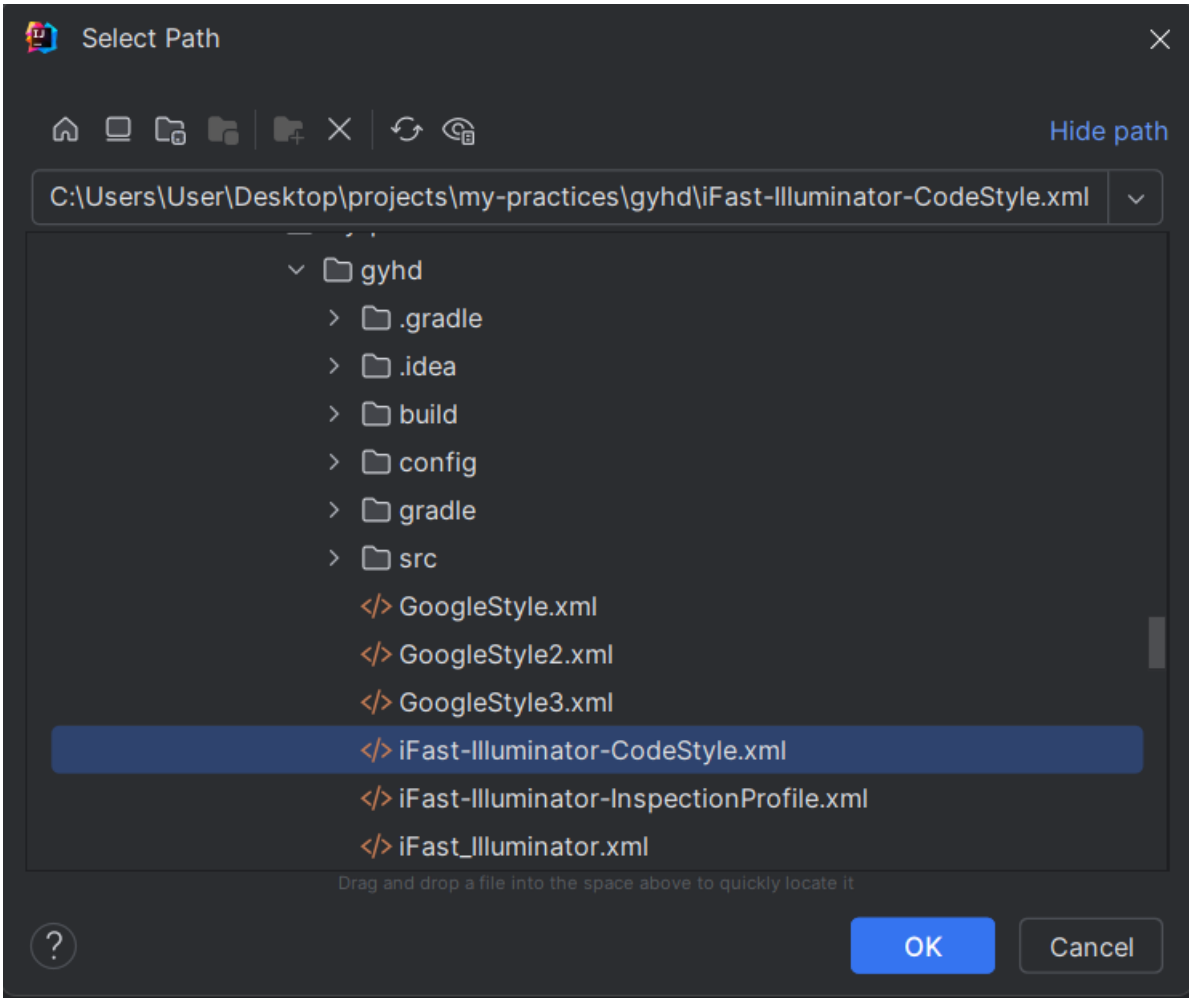
2. Open **Editor > Code Style > Java**



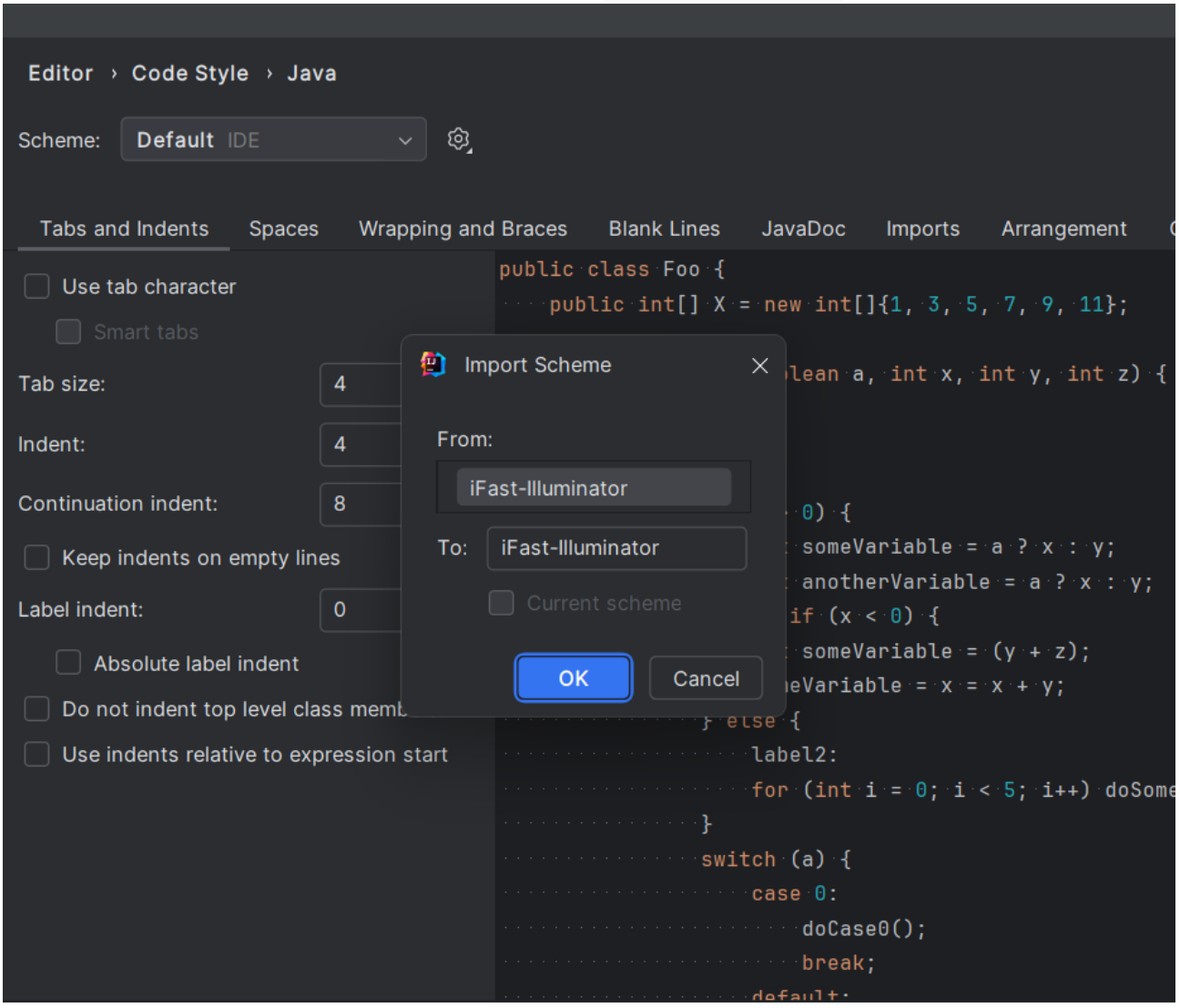
3. Click the Scheme dropdown (gear icon) > **Import Scheme > IntelliJ IDEA code style XML**



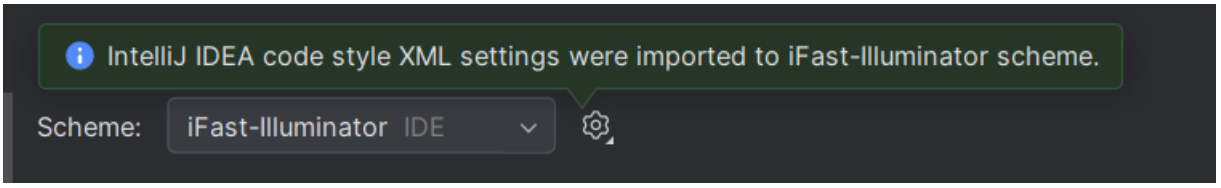
4. Select `iFast-Illuminator-CodeStyle.xml`. The scheme will load completely — the file's location won't affect future behavior.



5. Name the scheme (or keep the default: `iFast-Illuminator`).

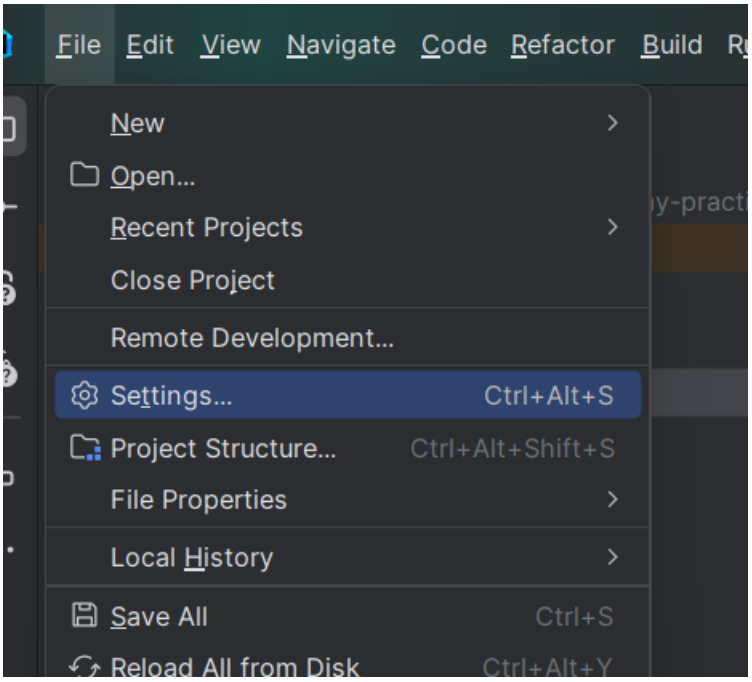


6. You should see the message:
“IntelliJ IDEA code style XML settings were imported to `<Your-Scheme-Name>`”
The scheme is now applied.

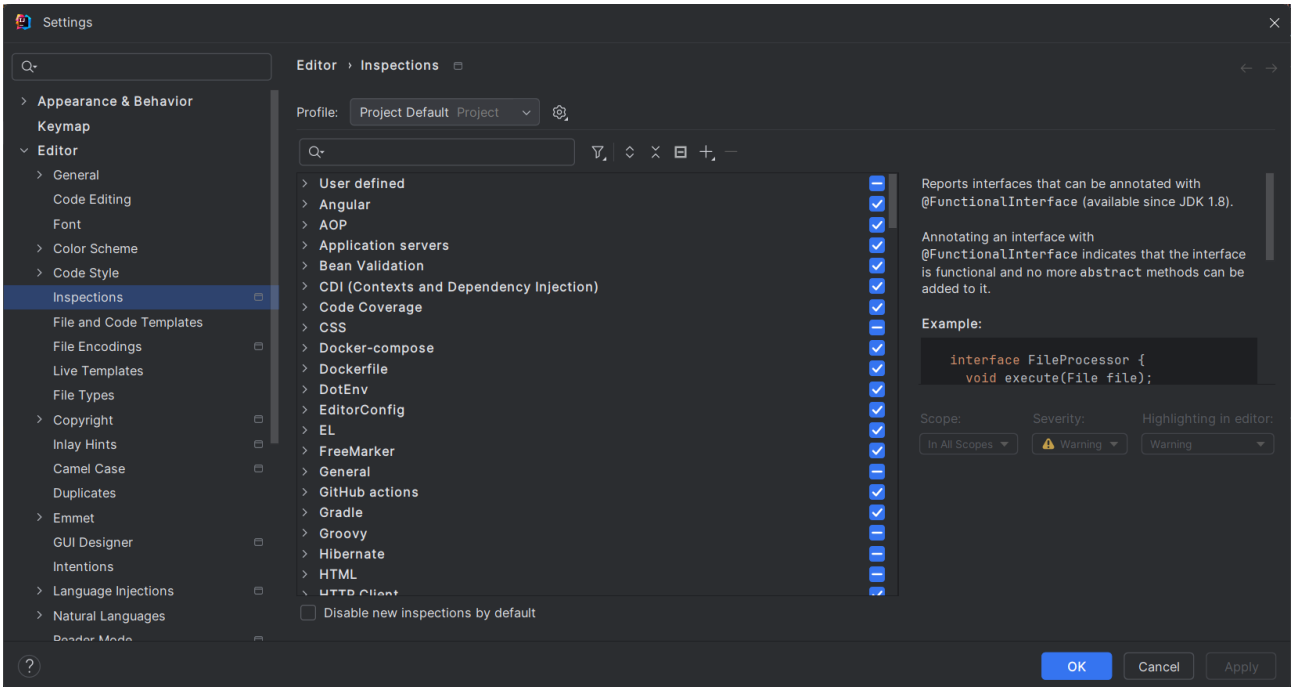


1.2.2. `iFast-Illuminator-InspectionProfile.xml`

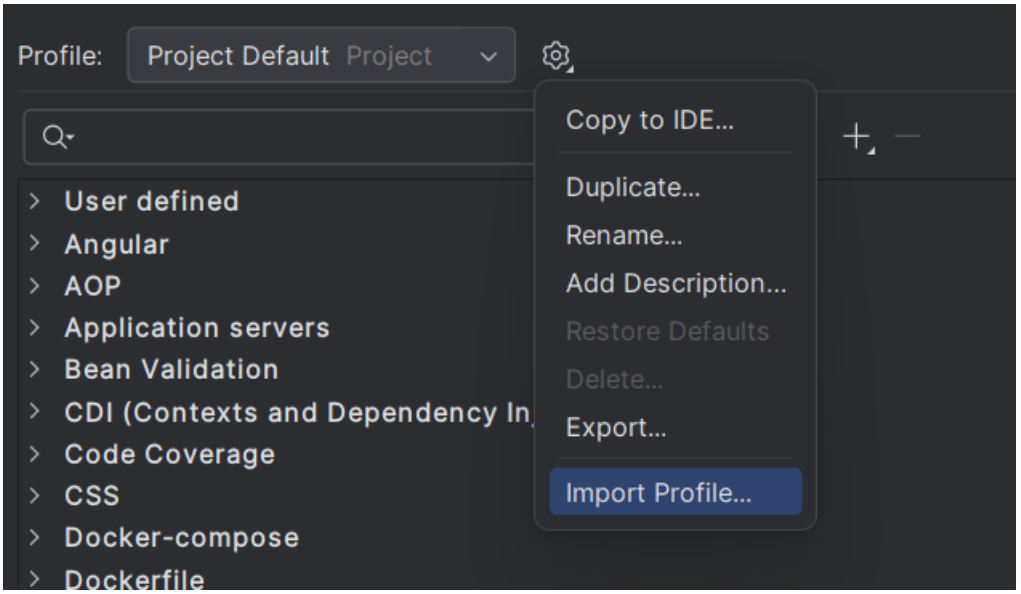
1. Open **File > Settings**



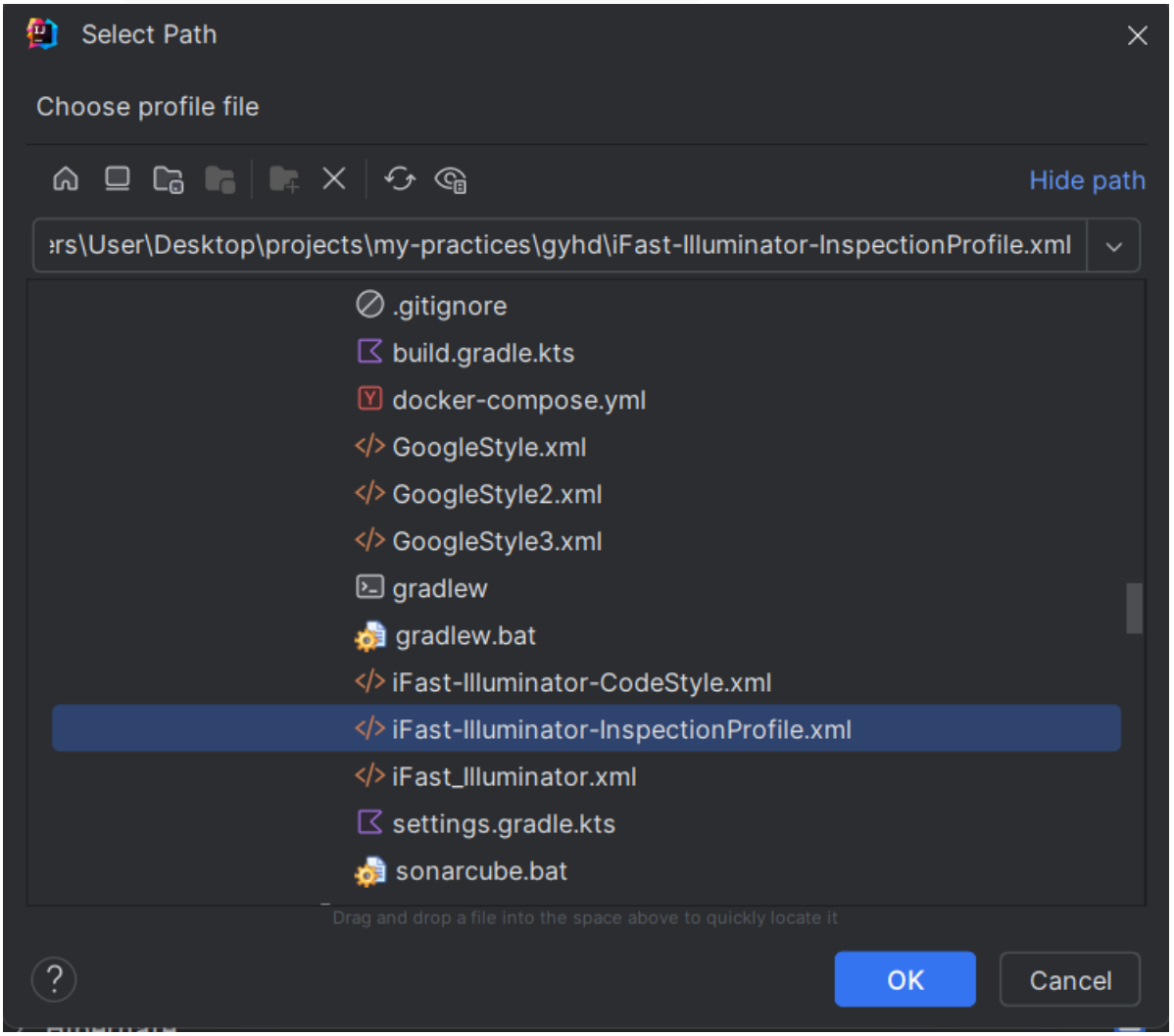
2. Open **Editor > Inspection**



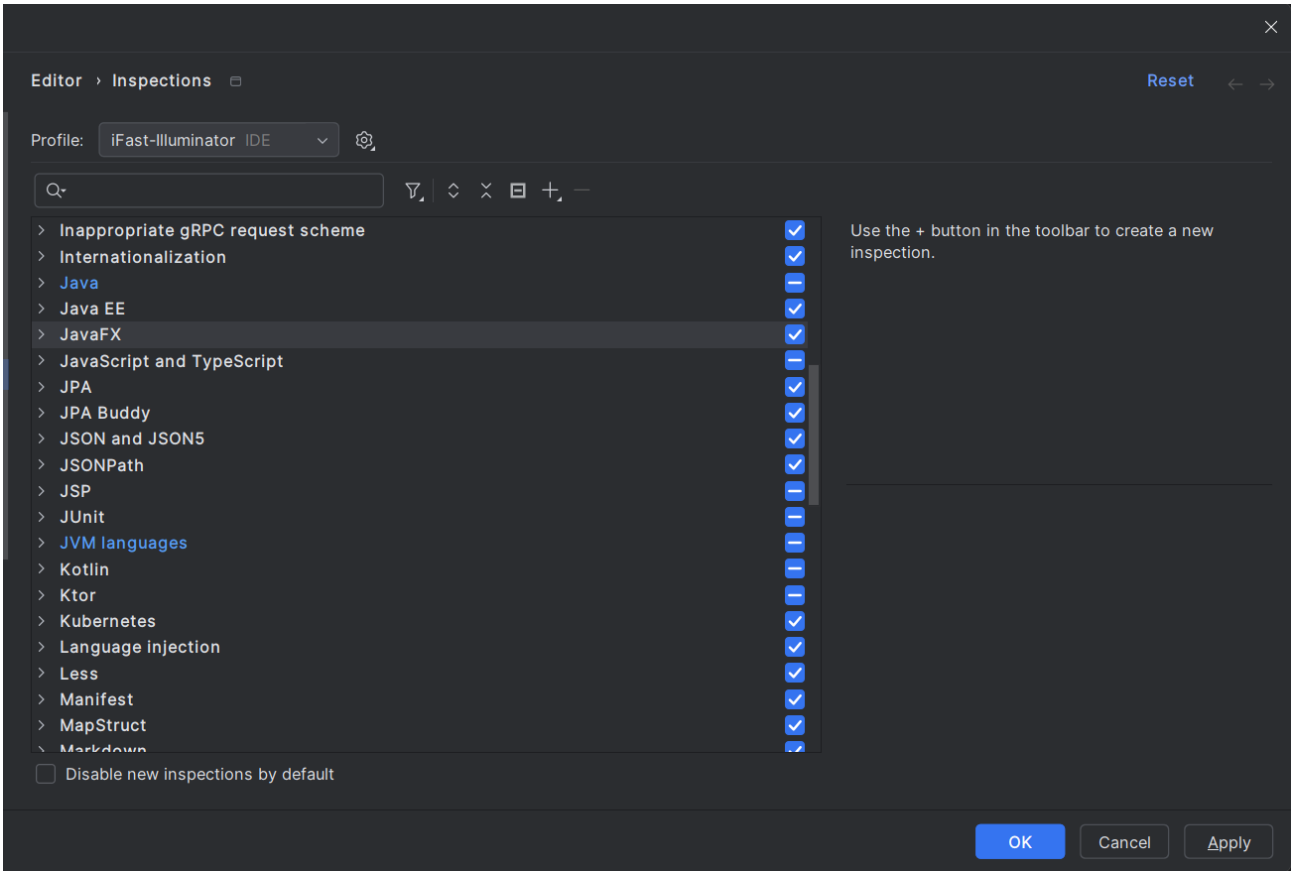
3. Click the Profile dropdown (gear icon) > **Import Profile**



4. Select `iFast-Illuminator-InspectionProfile.xml`



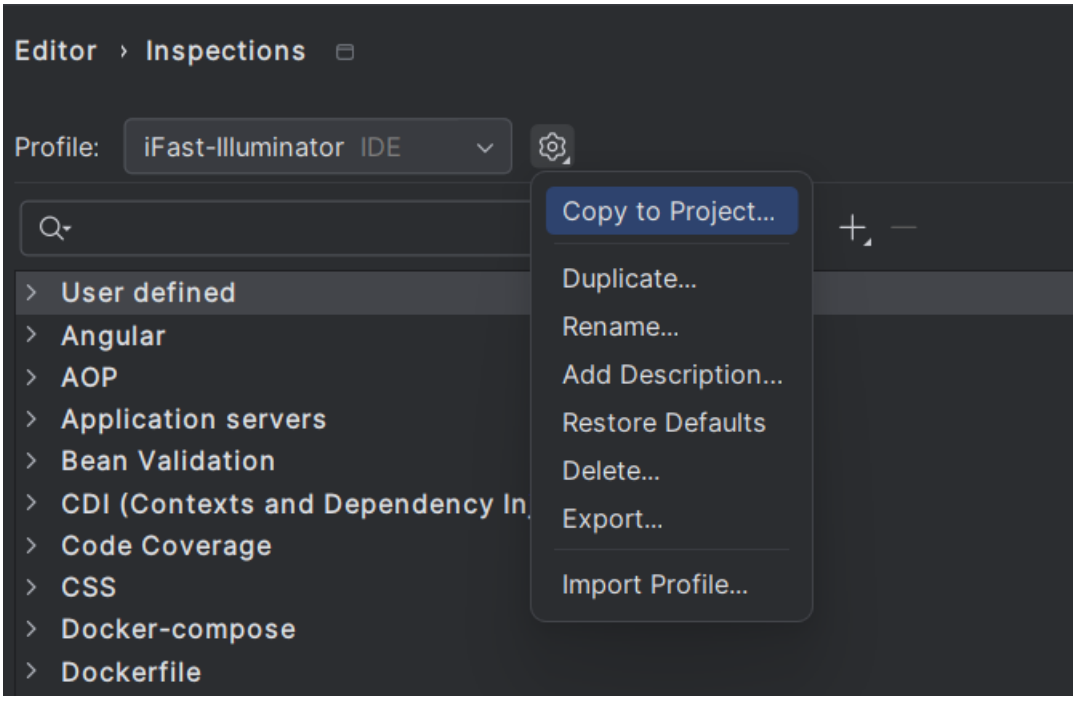
5. You should see the profile named `iFast-Illuminator`.
Confirm that the "Java" and "JVM languages" categories are highlighted in blue — this means the configuration is applied.



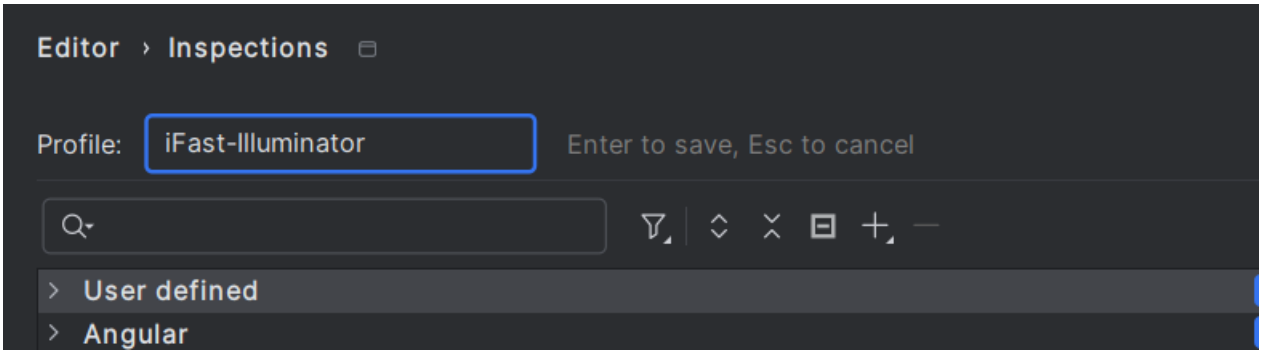
6. Click **OK.**
7. Go to `.idea/inspectionProfiles/profiles_settings.xml` to verify the profile was applied:

```
<component name="InspectionProjectProfileManager">
  <settings>
    <option name="PROJECT_PROFILE" value="iFast-Illuminator" />
    <option name="USE_PROJECT_PROFILE" value="false" />
    <version value="1.0" />
  </settings>
</component>
```

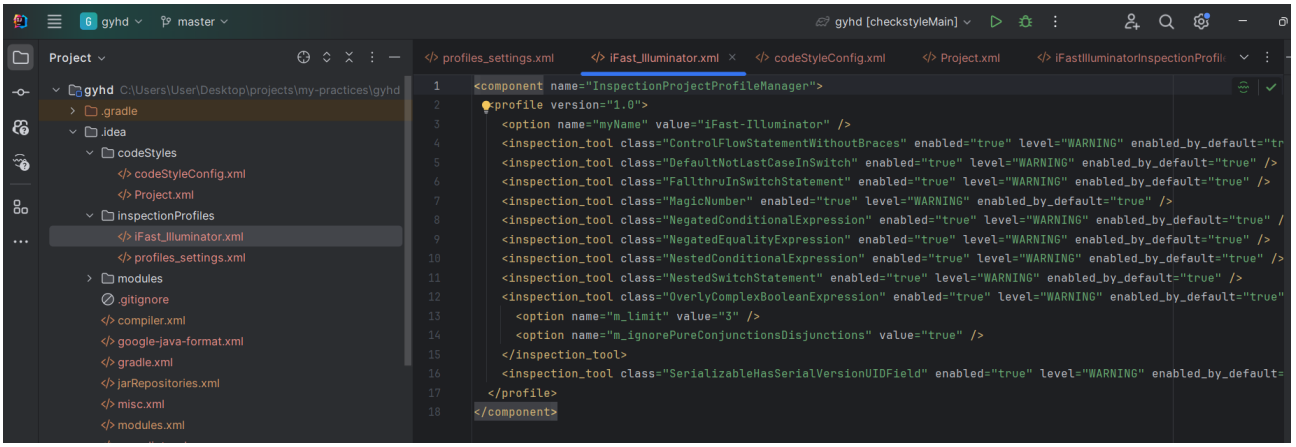
8. For further verification, in **Editor > Inspections > Profile**, click **Copy to Project**



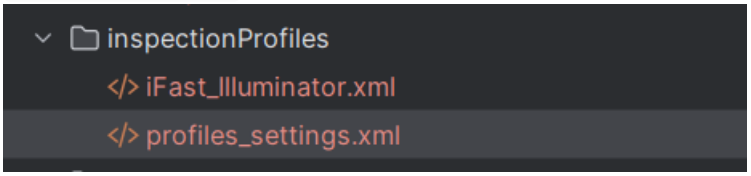
9. Keep the same name. This will copy the profile settings from IDE to the current project.



10. This will generate `.idea/inspectionProfiles/iFast_Illuminator.xml`. You can see the current configuration applied.



11. If using your own custom profile (e.g., `iFast-Illuminator`), you will see `profiles_settings.xml` — unless you are using the `Project Default` / delete the `Project Default`, in which case it won't appear.



1.3. Edit the configuration

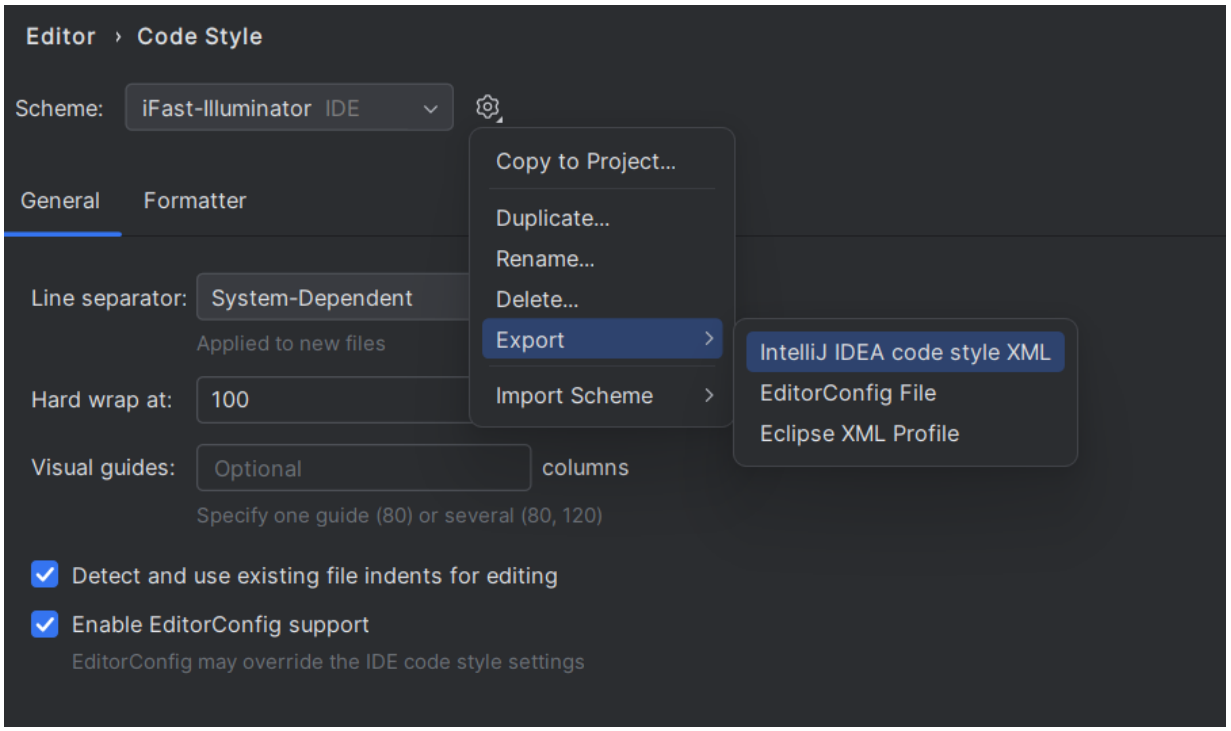
It is recommended to maintain these configuration files in a **centralized Git repository**. This way, when updates are made, developers can `git pull` to get the latest changes and import them effortlessly. Version control also enables rollback and collaborative improvements (via PRs/issues).

1.3.1. `iFast-Illuminator-CodeStyle.xml`

There are two approaches:

1. **(Less preferable yet easier)**
    1. Edit using IntelliJ, export to overwrite the previous scheme.
    2. ❗Be cautious — this WILL overwrite useful comments and default-value options that were purposefully included. (FYI, the IDE won't show

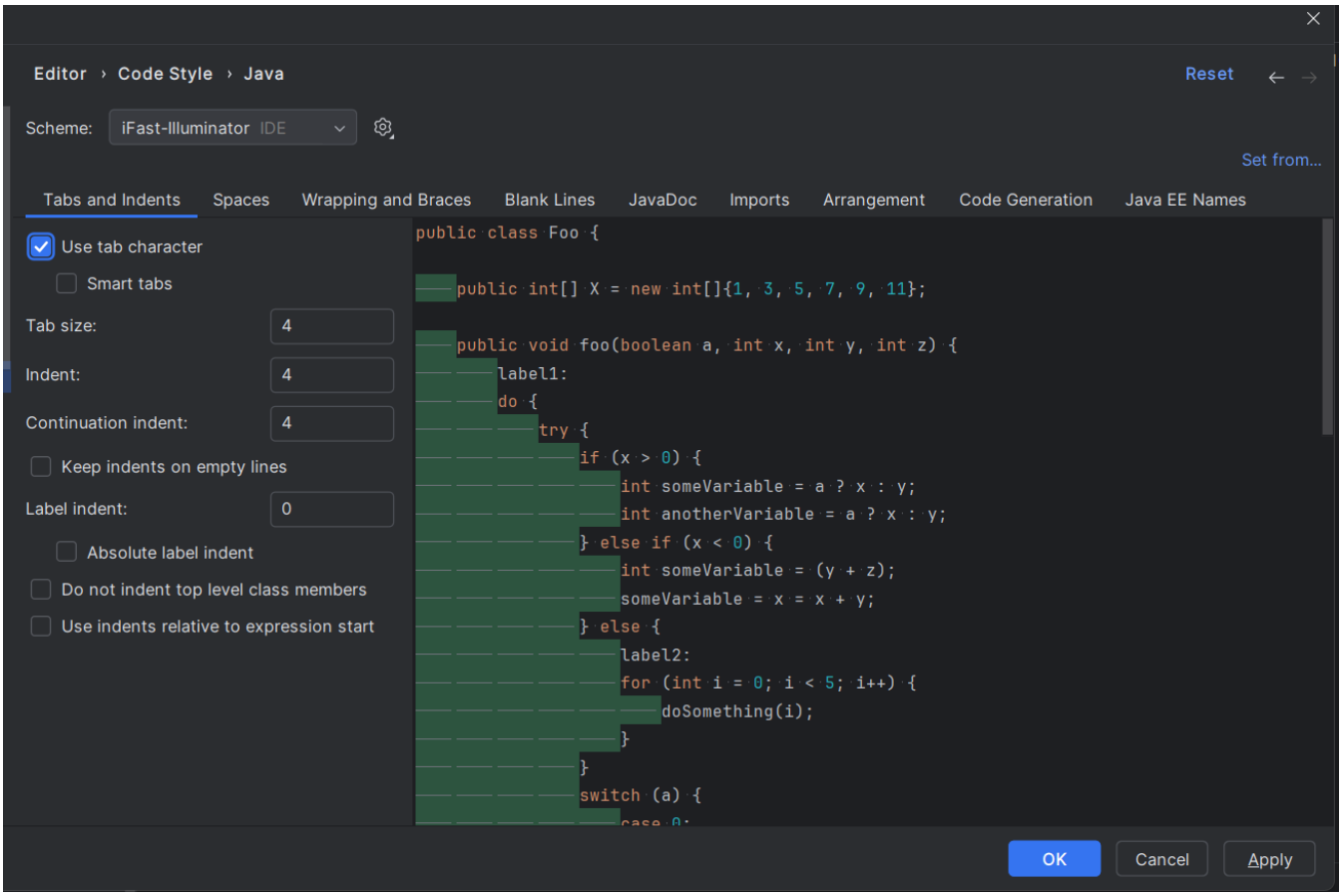
default-value options in the exported schema).



2. **(Preferable yet manual)**
    1. Export changes as a new file, then manually merge the changes into the team version (e.g., `iFast-Illuminator-CodeStyle.xml`).
    2. ✅ Preserves structure, comments, and intent behind the configuration.

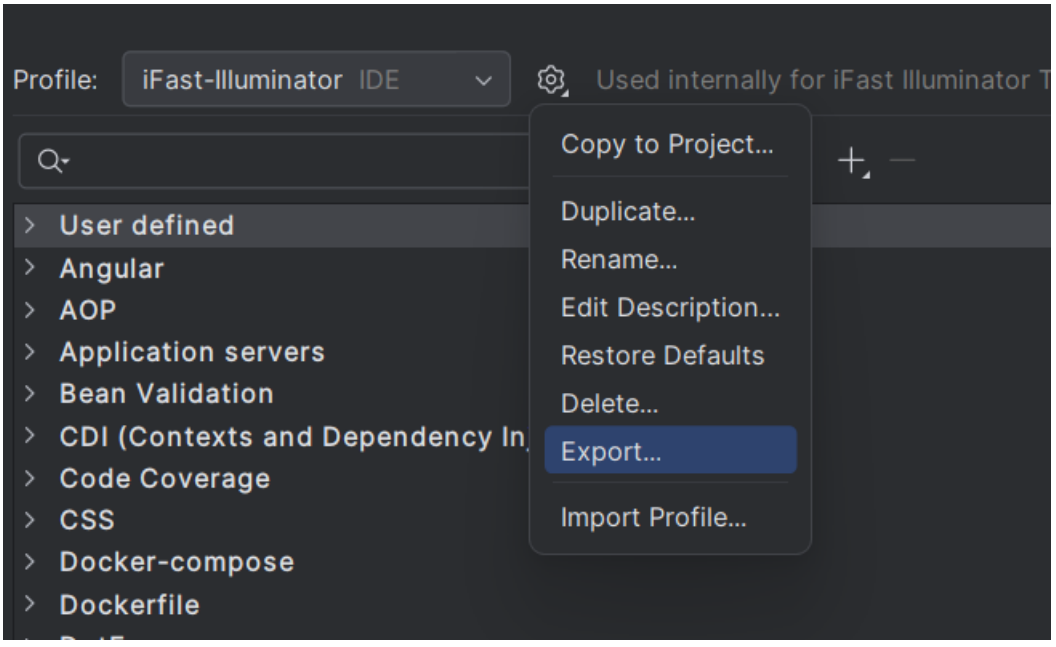
REMINDER:
ALWAYS click **Apply** or **OK** to apply the changes before exporting — otherwise, they won't be included in the saved schema.

- **Apply**: Saves changes and keeps the settings window open
- **OK**: Saves changes and closes the settings window



1.3.2. `iFast-Illuminator-InspectionProfile.xml`

This file is easier to manage. You can directly export and overwrite the old profile without losing important settings – since inspection tools are defined by class names (`class="..."`), the structure is self-explanatory (Unless you have added any comments).



1.4. FAQ

How do I use an updated profile that someone else modified?

You'll need to re-import the updated `.xml` schema. Importing is required each time changes are made by others.

Can I use `.editorconfig` and `.xml` files together?

Yes. IntelliJ uses configuration in this order:

1. `.editorconfig` (highest precedence)
2. XML configuration (e.g., `CodeStyle.xml`)
3. IDE defaults (lowest precedence)

However, we currently do **not** use `.editorconfig`.

Will autoformatting affect all files in the project?

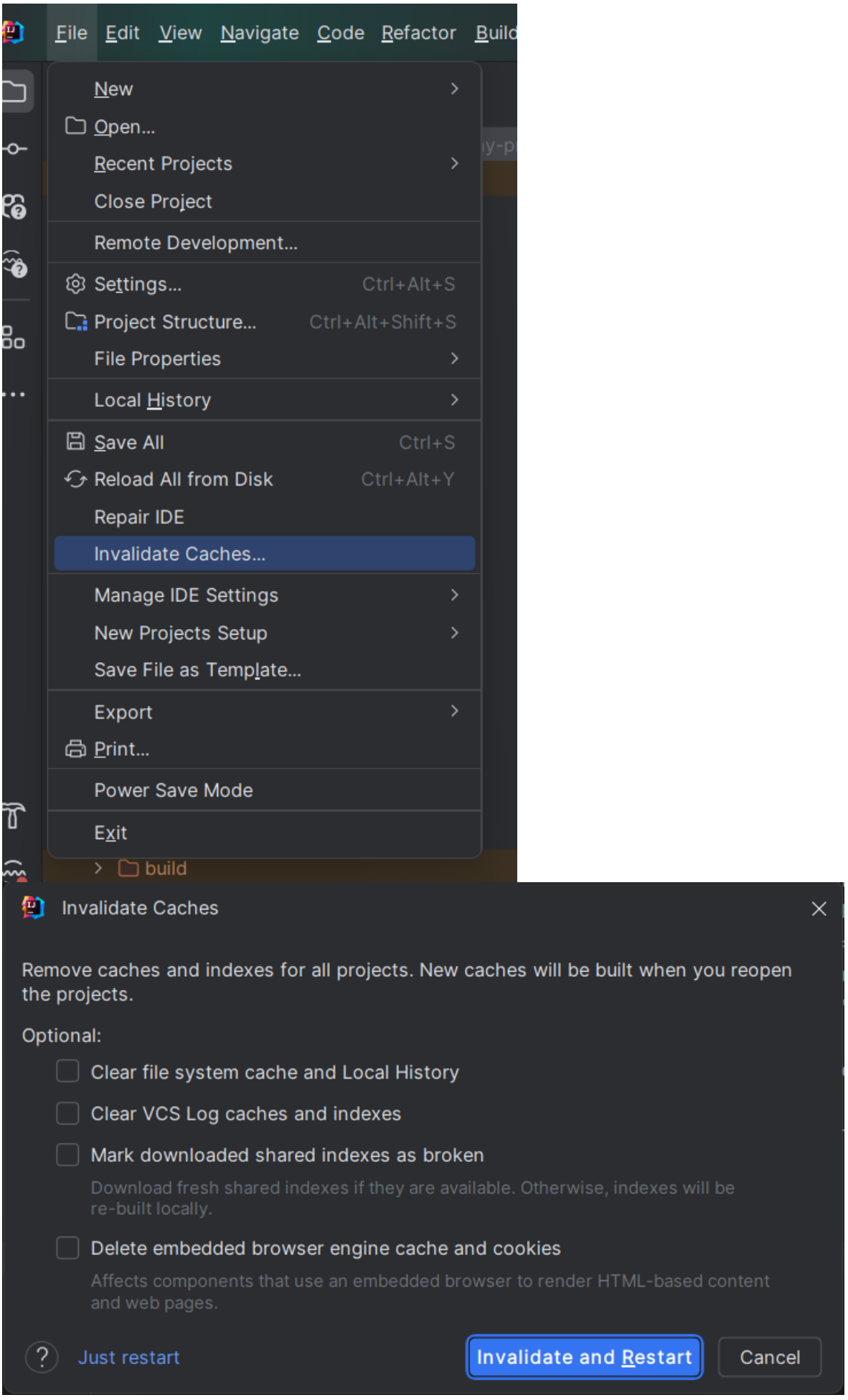
No. IntelliJ IDEA applies reformatting only to:

- Files you're editing
- Files saved, if Actions on Save is enabled
- Files explicitly selected during a bulk reformatting action

Why don't my changes appear?

Checklist:

- Make sure you clicked **Apply** or **OK** before exiting settings.
- If someone else updated the profile/schema, re-import the `.xml` file.
- If everything seems correct but still doesn't apply, restart the IDE: Go to **File > Invalidate Caches > Invalidate and Restart.**



2. (Optional) XML

You can found the complete files in the folder. This section is mainly used to describe some of the fields.

2.1. `IMPORT_LAYOUT_TABLE`

Before:

```
import java.util.List;
import static java.lang.Math.max;
import java.io.File;
import static java.lang.System.out;
```

After:

```
import static java.lang.Math.max;
import static java.lang.System.out;

import java.io.File;
import java.util.List;
```

2.2. `KEEP_BLANK_LINES_BEFORE_RBRACE=0`

Before:

```
public class Student {
    public static void main(String[] args) {
    }
    // There's a lot of blank lines here...



}
```

After:

```
public class Student {
    public static void main(String[] args) {
    }
    // There's no blank lines right now!
}
```

2.3. `BLANK_LINES_AFTER_CLASS_HEADER=1`

Before:

```
public class Student {




    // There's a lot of blank lines after class header.
    public static void main(String[] args) {
    }
}
```

After:

```
public class Student {

    // There's only one blank lines after class header right now!
    public static void main(String[] args) {
    }
}
```

2.4. `KEEP_BLANK_LINES_IN_CODE=1`

Before:

```
public class Student {
    public static void main(String[] args) {
        String name = "Student";



        int age = 25;



        double cgpa = 4.0;


    }
}
```

After:

```
public class Student {
    public static void main(String[] args) {
        String name = "Student";

        int age = 25;

        double cgpa = 4.0;
```

```
    }
}
```

2.5. ALIGN_MULTILINE_PARAMETERS

```
<option name="ALIGN_MULTILINE_PARAMETERS" value="true"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces >

Original:

```
private static <T extends Comparable<T>, Y> void longMethod(Supplier<T> supplierForT, Supplier<Y> supplierForY, Consumer<T> consumerForT) {

}
```

`true` :

```
private static <T extends Comparable<T>, Y> void longMethod(Supplier<T> supplierForT,
                                                            Supplier<Y> supplierForY,
                                                            Consumer<T> consumerForT) {


}
```

`false` :

```
private static <T extends Comparable<T>, Y> void longMethod(Supplier<T> supplierForT,
    Supplier<Y> supplierForY, Consumer<T> consumerForT) {

}
```

In both scenario, the IDE will split the parameters automatically after they exceed the maximum length allowed.

2.6. ALIGN_MULTILINE_RESOURCES

```
<option name="ALIGN_MULTILINE_RESOURCES" value="false"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces >
``

value=false:

```
try (Scanner sc1 = new Scanner(System.in);
     Scanner sc2 = new Scanner(System.in);
     Scanner sc3 = new Scanner(System.in)) {

}
```

value=true:

```
try (Scanner sc1 = new Scanner(System.in);
     Scanner sc2 = new Scanner(System.in);
     Scanner sc3 = new Scanner(System.in)) {

}
```

2.7. TERNARY_OPERATION_SIGNS_ON_NEXT_LINE

```
<option name="TERNARY_OPERATION_SIGNS_ON_NEXT_LINE" value="true"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Ternary operation

value=0, Do not wrap

```
    int y =
            2 > 3 ? 7 + 8 + 9 : 11
                    + 12 + 13;
```

(Currently in use) value=1, Wrap if long

```
  int y = 2 > 3 ? 7 + 8 + 9
        : 11 + 12 + 13;
```

value=2, Chop down if long

```
  int y = 2 > 3
      ? 7 + 8 + 9
      : 11 + 12 + 13;
```

value=3, Wrap always

```
  int y = 2 > 3
      ? 7 + 8 + 9
      : 11 + 12 + 13;
```

2.8. USE_RELATIVE_INDENTS

```
<option name="USE_RELATIVE_INDENTS" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Tabs and Indents

Will affect a lot of indentation*

false:

```
List<String> strings = students.stream() Stream<Student>
    .map(Student::toString) Stream<String>
    .sorted(Comparator.naturalOrder())
    .toList();
```

```
boolean complexCondition = 1 == 2
    && 2 == 3
    && 3 == 4;
```

true:

```
List<String> strings = students.stream() Stream<Student>
                        .map(Student::toString) Stream<String>
                        .sorted(Comparator.naturalOrder())
                        .toList();
```

```
boolean complexCondition = 1 == 2
                        && 2 == 3
                        && 3 == 4;
```

2.9. ALIGN_MULTILINE_CHAINED_METHODS

```
<option name="ALIGN_MULTILINE_CHAINED_METHODS" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

```
List<String> strings = students.stream() Stream<Student>
                        .map(Student::toString) Stream<String>
                        .sorted(Comparator.naturalOrder())
                        .toList();
```

```
List<String> strings = students.stream() Stream<Student>
    .map(Student::toString) Stream<String>
    .sorted(Comparator.naturalOrder())
    .toList();
```

2.10. ALIGN_MULTILINE_BINARY_OPERATION

```
<option name="ALIGN_MULTILINE_BINARY_OPERATION" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

Without any settings

```
int veryComplexIntegerComputation = 20;
int anotherVeryComplexIntegerComputation = 30;
int superComplexIntegerComputation = 40;

int sum = veryComplexIntegerComputation + 2
    + anotherVeryComplexIntegerComputation + 4
    + superComplexIntegerComputation;
```

With `relative indentation`

```
int veryComplexIntegerComputation = 20;
int anotherVeryComplexIntegerComputation = 30;
int superComplexIntegerComputation = 40;

int sum = veryComplexIntegerComputation + 2
              + anotherVeryComplexIntegerComputation + 4
              + superComplexIntegerComputation;
```

With `relative indentation` + `Binary Multiline Alignment` (The latter will overwrite)

```
int veryComplexIntegerComputation = 20;
int anotherVeryComplexIntegerComputation = 30;
int superComplexIntegerComputation = 40;

int sum = veryComplexIntegerComputation + 2
          + anotherVeryComplexIntegerComputation + 4
          + superComplexIntegerComputation;
```

2.11. ALIGN_MULTILINE_TERNARY_OPERATION

```
<option name="ALIGN_MULTILINE_TERNARY_OPERATION" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

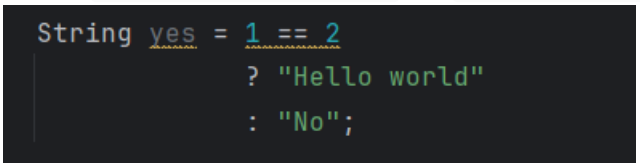
Without any settings

```
String yes = 1 == 2
    ? "Hello world"
    : "No";
```

With relative indentation

```
String yes = 1 == 2
                 ? "Hello world"
                 : "No";
```

With `relative indentation` + `Binary Multiline Alignment` (The latter will overwrite)

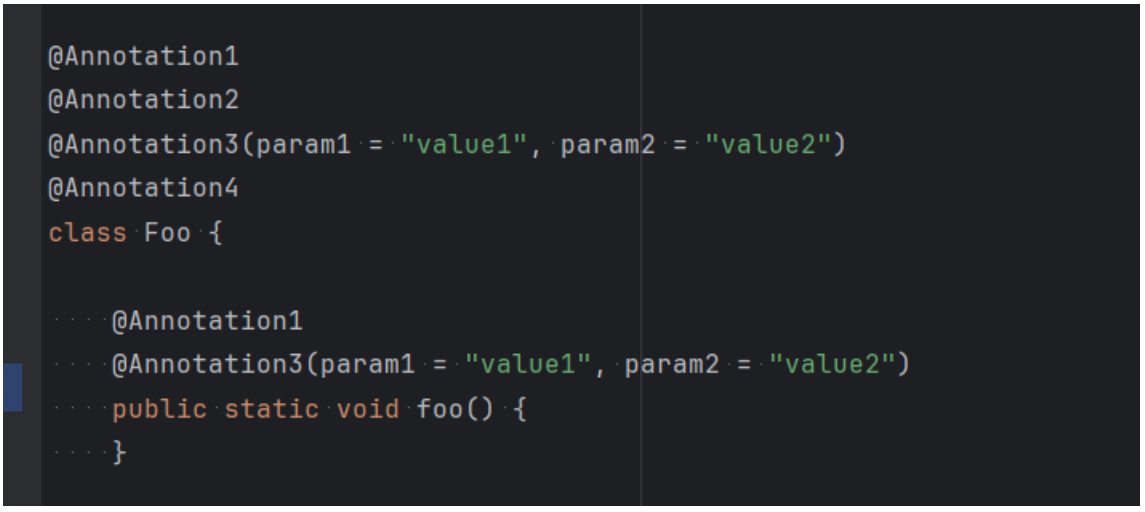


2.12. Annotation

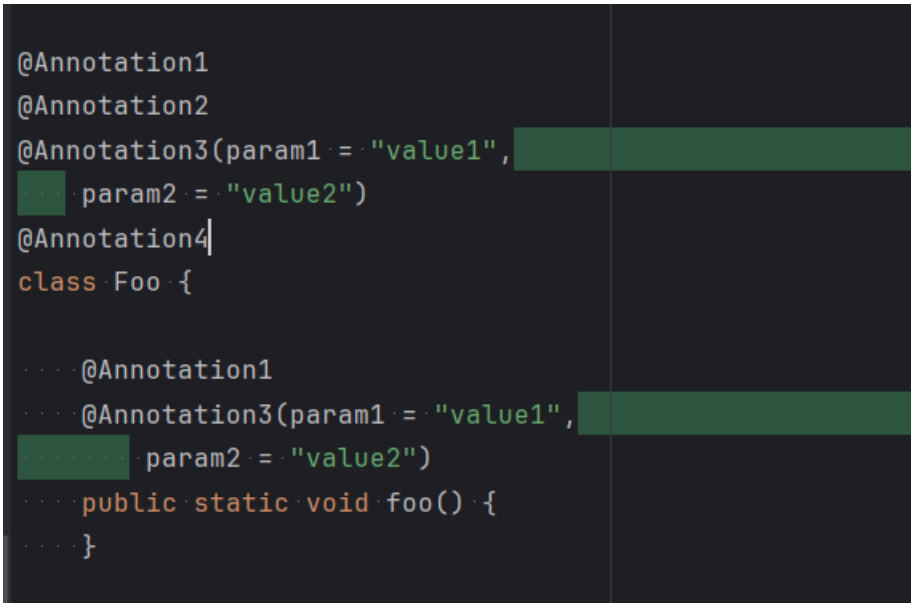
```
<!-- Debatable -->
<!-- Means the annotation will wrap parameters if they are too long -->
<option name="ANNOTATION_PARAMETER_WRAP" value="1" />

<!-- Debatable -->
<!-- Means the parameters will be aligned -->
<option name="ALIGN_MULTILINE_ANNOTATION_PARAMETERS" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

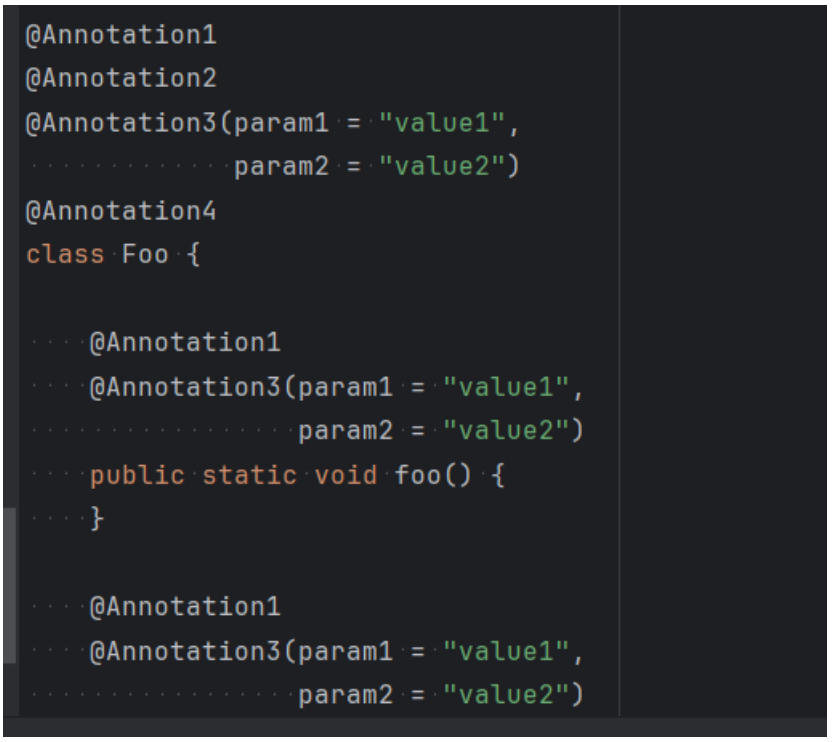
Without any config



Wrap if Long (value=1)



Align when multiline



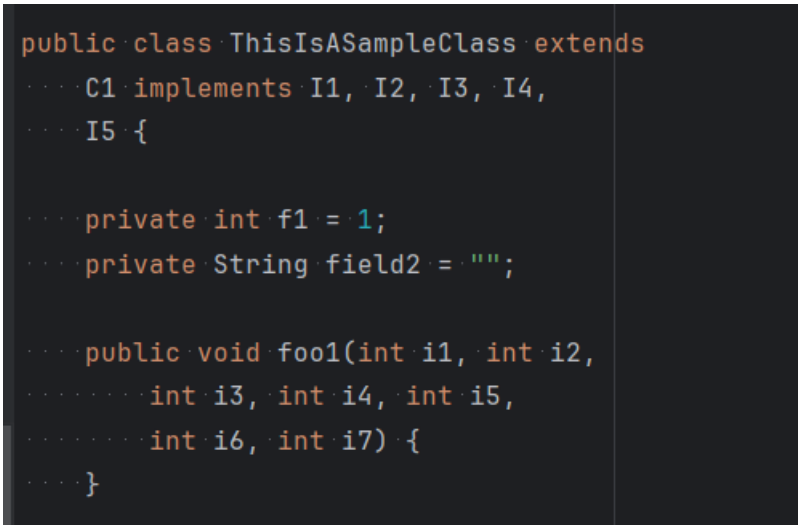
```
@Annotation1
@Annotation2
@Annotation3(param1 = "value1",
             param2 = "value2")
@Annotation4
class Foo {

    @Annotation1
    @Annotation3(param1 = "value1",
                 param2 = "value2")
    public static void foo() {
    }

    @Annotation1
    @Annotation3(param1 = "value1",
                 param2 = "value2")
```

2.13. ALIGN_GROUP_FIELD_DECLARATIONS

```
<option name="ALIGN_GROUP_FIELD_DECLARATIONS" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

Before:



```
public class ThisIsASampleClass extends
    C1 implements I1, I2, I3, I4,
    I5 {

    private int f1 = 1;
    private String field2 = "";

    public void foo1(int i1, int i2,
            int i3, int i4, int i5,
            int i6, int i7) {
    }
```

After:

```
public class ThisIsASampleClass extends
    C1 implements I1, I2, I3, I4,
    I5 {

    private int    f1     = 1;
    private String field2 = "";

    public void foo1(int i1, int i2,
        int i3, int i4, int i5,
        int i6, int i7) {
    }
```

2.14. ALIGN_CONSECUTIVE_VARIABLE_DECLARATIONS

```
<option name="ALIGN_CONSECUTIVE_VARIABLE_DECLARATIONS" value="true" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

Before:

```
        i = 0;
    int[] a = new int[]{1, 2,
        0x0052, 0x0053, 0x0054};
    int[] empty = new int[]{};
    int var1 = 1;
    int var2 = 2;
    foo1(0x0051, 0x0052, 0x0053,
        0x0054, 0x0055, 0x0056,
        0x0057);
    int x =
        (3 + 4 + 5 + 6) * (7 + 8
            + 9 + 10) * (11 + 12
            + 13 + 14
            + 0xFFFFFFFF);
    String s1, s2, s3;
    s1 = s2 = s3 = "012345678901456";
```

After:

```
        i = 0;
    int[] a      = new int[]{1, 2,
        0x0052, 0x0053, 0x0054};
    int[] empty = new int[]{};
    int    var1  = 1;
    int    var2  = 2;
    foo1(0x0051, 0x0052, 0x0053,
        0x0054, 0x0055, 0x0056,
        0x0057);
    int    x =
        (3 + 4 + 5 + 6) * (7 + 8
            + 9 + 10) * (11 + 12
            + 13 + 14
            + 0xFFFFFFFF);
```

2.15. Throw

```
<option name="THROWS_LIST_WRAP" value="0" />
<option name="ALIGN_MULTILINE_THROWS_LIST" value="false" />
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

Without any settings

```
public static void longerMethod()
    throws Exception1, Exception2, Exception3 {
```

With Wrap if Long (value=1)

```
public static void longerMethod()
    throws Exception1,
    Exception2, Exception3 {
```

With Wrap if Long (value=1) + Align when multiline

```
public static void longerMethod()
    throws Exception1,
           Exception2,
           Exception3 {
```

2.16. Records

```
<option name="RECORD_COMPONENTS_WRAP" value="0" />
<option name="ALIGN_MULTILINE_RECORDS" value="false" />
<option name="NEW_LINE_AFTER_LPAREN_IN_RECORD_HEADER" value="false" />
<option name="RPAREN_ON_NEW_LINE_IN_RECORD_HEADER" value="false" />
```

Without any settings

```
public record Student(String someVeryLongNameHereToDisplay,
    String anotherVeryLongEmailToDisplay) {  no usages

}
```

With `RECORD_COMPONENTS_WRAP` / Manual line breaks + `ALIGN_MULTILINE_RECORDS`

```
public record Student(String someVeryLongNameHereToDisplay,  6
                      Rename usages
                      String anotherVeryLongEmailToDisplay) {

}
```

```
public record Student(  6 usages  new *
    String someVeryLongNameHereToDisplay,  no usages
    String anotherVeryLongEmailToDisplay) {  no usages

}
```

With all options

```
public record Student(  6 usages  new *
    String someVeryLongNameHereToDisplay,  no usages
    String anotherVeryLongEmailToDisplay  no usages
) {

}
```

> **Info**
>
> Basically, by turning on all the options, it will automatically turn it into the last format. Without `wrap_if_long`, the developers would need to insert the line break manually.

2.17. Control Statement

- `KEEP_CONTROL_STATEMENT_IN_ONE_LINE`
- `IF_BRACE_FORCE`

```
<option name="KEEP_CONTROL_STATEMENT_IN_ONE_LINE" value="false"/>
<option name="IF_BRACE_FORCE" value="3"/>
```

This configuration will automatically add braces for if-else statements and force them to be in multiline.

Before:

```
if (2 == 3) System.out.println("Yes"); else System.out.println("No");
```

After:

```
if (2 == 3) {
    System.out.println("Yes");
} else {
    System.out.println("No");
}
```

2.18. KEEP_BLANK_LINES_IN_CODE

```
<option name="KEEP_BLANK_LINES_IN_CODE" value="1"/>
```

This option will keep only 1 blank line between code section.

Before:

```
boolean complexCondition = 1 == 2 && 2 == 3 && 3 == 4;
String cond = """
    Hello world
    Lets go
    """;



System.out.println(cond);
```

After:

```
boolean complexCondition = 1 == 2 && 2 == 3 && 3 == 4;
String cond = """
    Hello world
    Lets go
    """;

System.out.println(cond);
```

2.19. BLANK_LINES_AFTER_CLASS_HEADER

```
<option name="BLANK_LINES_AFTER_CLASS_HEADER" value="1"/>
```

This option will add a blank line after class header.

Before:

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

After:

```
public class Main {

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

2.20. METHOD_PARAMETERS_WRAP

```
<option name="METHOD_PARAMETERS_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Method declaration parameters

value=0, Do not wrap

```
public void foo1(int i1, int i2, int i3, int i4, int i5, int i6, int i7)
}
```

(Currently in use) value=1, Wrap if long

```
public void foo1(int i1, int i2,
    int i3, int i4, int i5,
    int i6, int i7) {
}
```

value=2, Chop down if long

```
public void foo1(int i1,
        int i2,
        int i3,
        int i4,
        int i5,
        int i6,
        int i7) {
}
```

2.21. EXTENDS_LIST_WRAP

```
<option name="EXTENDS_LIST_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Extends/implements/permits list

value=0, Do not wrap

```
public class ThisIsASampleClass extends C1 implements I1, I2, I3, I4, I5 {
```

(Currently in Use) value=1, Wrap if Long

```
public class ThisIsASampleClass extends
    C1 implements I1, I2, I3, I4,
    I5 {
```

2.22. THROWS_KEYWORD_WRAP

```
<option name="THROWS_KEYWORD_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Throws keyword

value=0, Do not wrap

```
  public static void longerMethod() throws Exception1, Exception2, Excepti
```

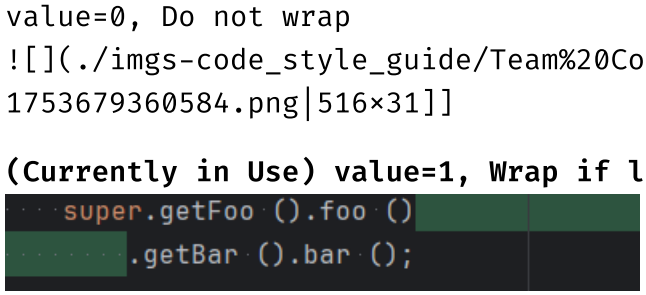
(Currently in Use) value=1, Wrap if long

```
    public static void longerMethod()
            throws Exception1, Exception2, Exception3 {
```

2.23. METHOD_CALL_CHAIN_WRAP

```
<option name="METHOD_CALL_CHAIN_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Chained method calls

value=0, Do not wrap


(Currently in Use) value=1, Wrap if long

```
    super.getFoo ().foo ()
            .getBar ().bar ();
```

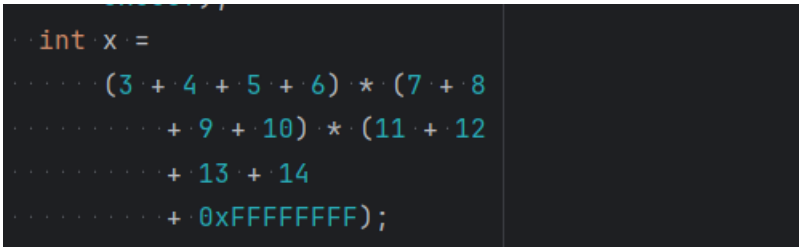
2.24. BINARY_OPERATION_WRAP

```
<option name="BINARY_OPERATION_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Binary expressions

value=0, Do not wrap

(Currently in Use) value=1, Wrap if long

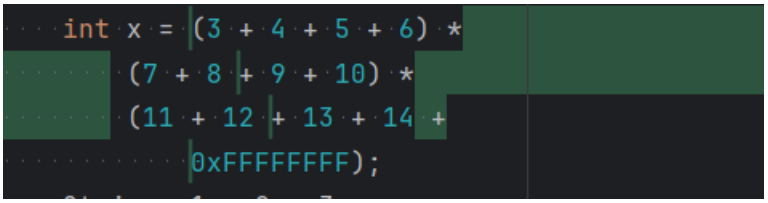


2.25. BINARY_OPERATION_SIGN_ON_NEXT_LINE

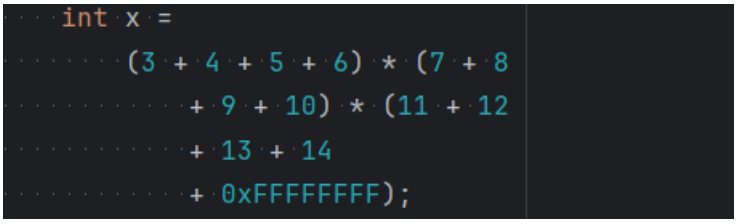
```
<option name="BINARY_OPERATION_SIGN_ON_NEXT_LINE" value="true"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Binary expressions > Operation sign on next line

value=false



(Currently in Use) value=true

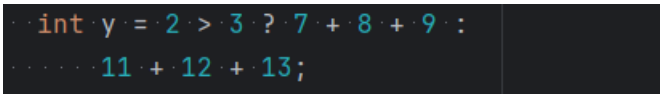


2.26. TERNARY_OPERATION_WRAP

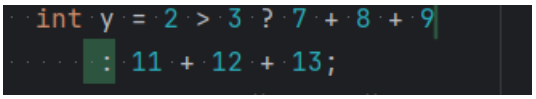
```
<option name="TERNARY_OPERATION_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Ternary operation > Operation sign on next line

value=false



(Currently in use) value=true

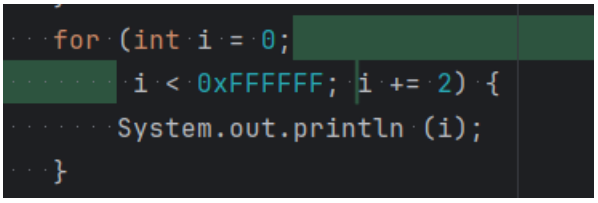


2.27. FOR_STATEMENT_WRAP

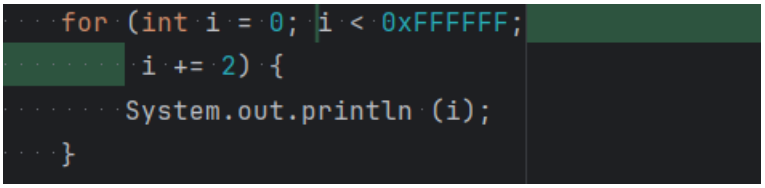
```
<option name="FOR_STATEMENT_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > `for()` statement

value=false, Do not wrap



(Currently in use) value=1, Wrap if long

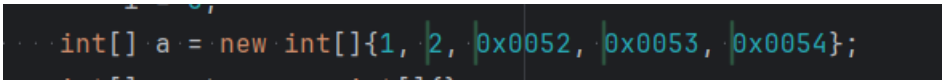


2.28. ARRAY_INITIALIZER_WRAP

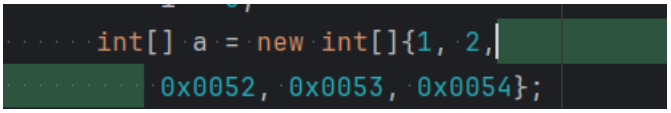
```
<option name="ARRAY_INITIALIZER_WRAP" value="1"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces > Array Initializer

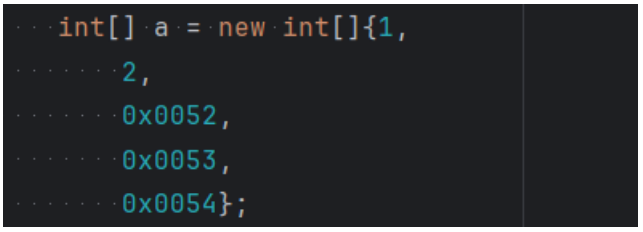
value=0, Do not wrap



**(Currently in use) value=1, Wrap if long



value=2, Chop down if long



2.29. Other braces

```
<option name="DOWHILE_BRACE_FORCE" value="3"/>
<option name="WHILE_BRACE_FORCE" value="3"/>
<option name="FOR_BRACE_FORCE" value="3"/>
```

Manual Config: Settings > Editor > Code Style > Java > Wrapping and Braces

value=3, (Force braces=Always, always add braces)

3. References

If you are using the `google-java-format` plugin, kindly refer to this guide. However, some configurations (e.g., indentation) provided by the Google Java Style may not align with the habits or preferences of the majority.