code-style-guide

Useful setup

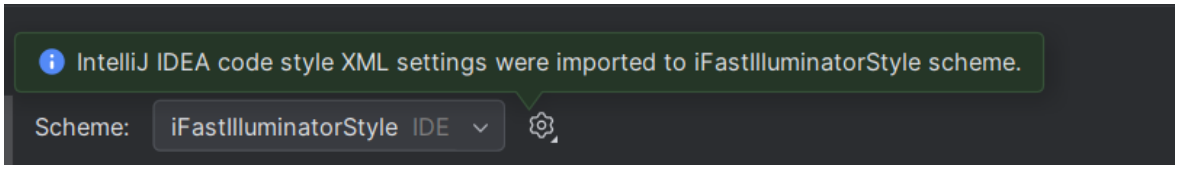
1. Settings > Tools > Actions on Save > Reformat Code, Optimize imports, Rearrange Code, Code Cleanup
2. Reformat File Dialog (Ctrl + Alt + Shift + L) > Reformat Code, Optimize imports, Rearrange Code, Code Cleanup
3. Settings > Appearance & Behavior > System Settings > Autosave > Save files if the IDE is idle for 60 seconds
4. Settings > Editor > General > Soft Wraps > Soft Wraps these files (It is helpful if you're using `Markdown`)
5. Settings > Languages & Frameworks > Markdown > Markdown Extensions (`PlantUML`) (If you're using `PlantUML`)

Suggestions

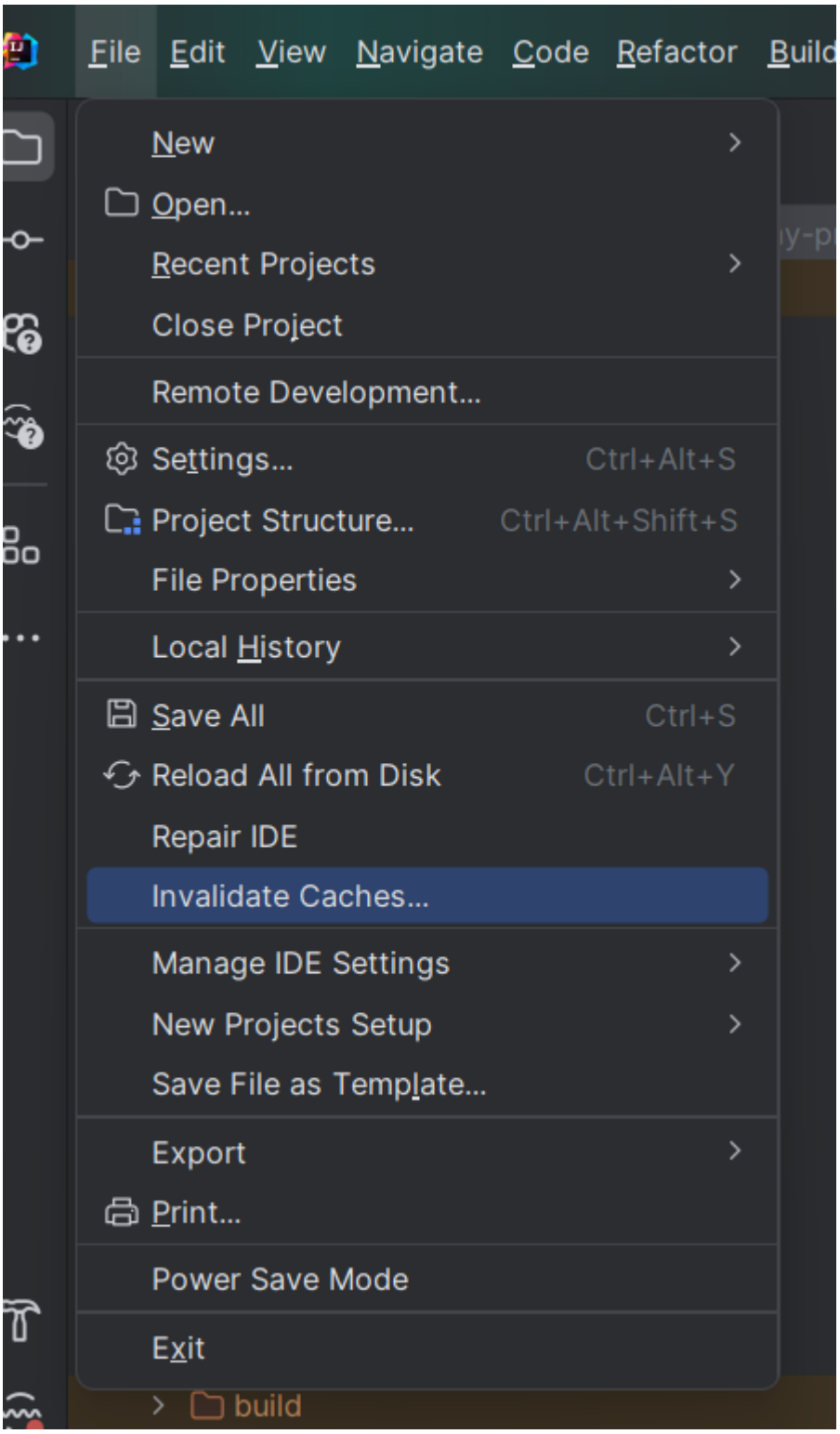
1. Use `google-java-format` plugin. Refer the official page for the setup: https://github.com/google/google-java-format/blob/master/README.md#intellij-jre-config.
2. Use the following IntelliJ-based solutions:
    1. `.editorconfig` / `ifastIlluminatorStyle.xml`: indentation, alignment, and chain wrapping
    2. `.idea/inspectionProfiles/Project_Default.xml`: For warning code smells, security issues, and potential bugs. Currently, it will warn on `ControlFlowStatementWithoutBraces`, `MagicNumber` and `SerializableHasSerialVersionUIDField`.
    3. `.idea/codeStyles/Project.xml`: To configure Java code rearranging properties. It will help you to rearrange the class fields according to the rules specified. Currently, it will rearrange in the following sequence: Service > Validator > Dao > Repository > Mapper.
3. Use `SonarCube IDE` plugin. `SonarCube` can be used for detecting code smells, security issues, and potential bugs. Advanced features (such as toggling on/off for the whole team) will require advanced setup.

FAQ

- Ques: Can I use XML + `.editorconfig` together?
- Ans: Yes. The IDE will use the configuration in `.editorconfig` first, then the customized XML, then finally the IDE default settings. However, the current XML configuration is sufficient enough to cover almost all the scenarios.
- Ques: How to use XML/ `.editorconfig` in my project?
- Ans: It depends
  - `.editorconfig`: Copy it directly it to the root folder. This approach is no longer in use.
  - `ifastIlluminatorStyle.xml`: Settings > Editor > Code Style > Java > Scheme (Settings icon) > Import Scheme > IntelliJ code style XML. It should show a message like this.

  - `.idea/inspectionProfiles/Project_Default.xml`: Copy the content of the file to the target file. If there's no, create one.
  - `.idea/codeStyles/Project.xml`: Copy the content of the file to the target file. If there's no, create one.
- Ques: Will autoformat affect my whole files? (Perhaps no)
- Ans: In IntelliJ IDEA, enabling automatic formatting on save **does not automatically reformat every file** in your project by default—it only affects files based on your settings and scope selections.
- Ques: Why it seems like the configuration didn't apply?
- Ans: The solutions depend on the your scenario:
  - For any changes in the `ifastIlluminatorStyle.xml`, you should reimport the scheme again to load the new changes.
  - For any changes in the `.editorconfig`, the changes should apply instantly.
  - If the configuration still didn't apply after all, invalidate the cache and restart the IDE. File (Menu bar) > Invalidate Caches



- [ ] Coding Style
  - [ ] How to use Checkstyle & specify the rules?
  - [ ] How to integrate Checkstyle into the gitlab cicd?
    - [ ] How to perform testing on latest CICD safely?
  - [ ] Is there any other tools to visualize Checkstyle (since currently it's merely command line tools)
  - [ ] What are the capabilities of Spotless?

1. XML-General

1.1. XML File Overview

```
<code_scheme name="GoogleStyle">
    <!-- Completely follow Google Java Format-->
    <option name="OTHER_INDENT_OPTIONS">
        <value>
            <option name="INDENT_SIZE" value="4"/>
            <option name="CONTINUATION_INDENT_SIZE" value="4"/>
            <option name="TAB_SIZE" value="2"/>
            <option name="USE_TAB_CHARACTER" value="false"/>
            <option name="SMART_TABS" value="false"/>
            <option name="LABEL_INDENT_SIZE" value="0"/>
            <option name="LABEL_INDENT_ABSOLUTE" value="false"/>
            <option name="USE_RELATIVE_INDENTS" value="false"/>
        </value>
    </option>

    <option name="RIGHT_MARGIN" value="100"/>
    <!-- Keep only 1 blank lines between code -->
    <option name="KEEP_BLANK_LINES_IN_CODE" value="1"/>
    <!-- Split control statement into multiple lines -->
    <option name="KEEP_CONTROL_STATEMENT_IN_ONE_LINE"
value="false"/>
    <!-- Remove any blank lines before the class end -->
    <option name="KEEP_BLANK_LINES_BEFORE_RBRACE" value="0"/>
    <option name="BLANK_LINES_AFTER_CLASS_HEADER" value="0"/>
    <option name="ALIGN_MULTILINE_PARAMETERS" value="false"/>
    <option name="ALIGN_MULTILINE_FOR" value="false"/>
    <option name="SPACE_BEFORE_ARRAY_INITIALIZER_LBRACE"
value="true"/>
    <!-- Wrap only if the code is too long -->
    <option name="CALL_PARAMETERS_WRAP" value="1"/>
    <option name="METHOD_PARAMETERS_WRAP" value="1"/>
    <option name="EXTENDS_LIST_WRAP" value="1"/>
    <option name="THROWS_KEYWORD_WRAP" value="1"/>
    <option name="METHOD_CALL_CHAIN_WRAP" value="1"/>
    <option name="BINARY_OPERATION_WRAP" value="1"/>
    <option name="BINARY_OPERATION_SIGN_ON_NEXT_LINE"
value="true"/>
    <option name="TERNARY_OPERATION_WRAP" value="1"/>
    <option name="TERNARY_OPERATION_SIGNS_ON_NEXT_LINE"
value="true"/>
```

```
    <!-- Forces to add braces even for one-liner -->
    <!-- Completely follow Google Java Format-->
    <option name="FOR_STATEMENT_WRAP" value="1"/>
    <option name="ARRAY_INITIALIZER_WRAP" value="1"/>
    <option name="WRAP_COMMENTS" value="true"/>
    <option name="IF_BRACE_FORCE" value="3"/>
    <option name="DOWHILE_BRACE_FORCE" value="3"/>
    <option name="WHILE_BRACE_FORCE" value="3"/>
    <option name="FOR_BRACE_FORCE" value="3"/>
    <JavaCodeStyleSettings>
        <!-- Debatable -->
        <option name="ANNOTATION_PARAMETER_WRAP" value="0" />
        <!-- Debatable -->
        <option name="ALIGN_MULTILINE_ANNOTATION_PARAMETERS"
value="false" />
        <!-- Debatable -->
        <option name="RECORD_COMPONENTS_WRAP" value="0" />
        <!-- Debatable -->
        <option name="ALIGN_MULTILINE_RECORDS" value="false" />
        <!-- Debatable -->
        <option name="NEW_LINE_AFTER_LPAREN_IN_RECORD_HEADER"
value="false" />
        <!-- Debatable -->
        <option name="RPAREN_ON_NEW_LINE_IN_RECORD_HEADER"
value="false" />

        <!-- Disable wildcard imports -->
        <!-- Completely follow Google Java Format-->
        <option name="INSERT_INNER_CLASS_IMPORTS" value="true"/>
        <option name="CLASS_COUNT_TO_USE_IMPORT_ON_DEMAND"
value="999"/>
        <option name="NAMES_COUNT_TO_USE_IMPORT_ON_DEMAND"
value="999"/>
        <option name="PACKAGES_TO_USE_IMPORT_ON_DEMAND">
            <value/>
        </option>
        <!-- Specify the layout describing how imports should be
organized  -->
        <!-- Completely follow Google Java Format -->
        <option name="IMPORT_LAYOUT_TABLE">
            <value>
                <package name="" withSubpackages="true"
static="true"/>
                <emptyLine/>
```

```
                <package name="" withSubpackages="true"
static="false"/>
            </value>
        </option>
        <!-- Specify the Javadocs format  -->
        <!-- Mostly follow Google Java Format -->
        <!-- Settings > Editor > Code Style > Java > JavaDoc -->
        <option name="JD_ALIGN_PARAM_COMMENTS" value="false"/>
        <option name="JD_ALIGN_EXCEPTION_COMMENTS" value="false"/>
        <option name="JD_P_AT_EMPTY_LINES" value="false"/>
        <option name="JD_KEEP_INVALID_TAGS" value="false"/>
        <option name="JD_KEEP_EMPTY_PARAMETER" value="false"/>
        <option name="JD_KEEP_EMPTY_EXCEPTION" value="false"/>
        <option name="JD_KEEP_EMPTY_RETURN" value="false"/>
        <option name="JD_PRESERVE_LINE_FEEDS" value="true"/>
    </JavaCodeStyleSettings>
</code_scheme>
```

1.2. Explanation

Note: Some other options are explained in the `XML - Java section`.

1.2.1. `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;
```

1.2.2. `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!
}
```

1.2.3. `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) {
    }
}
```

1.2.4. `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. XML - Java

```
<!-- Controversial options will be pointed out only -->
<!-- Others will just follow Google Java Format -->
<codeStyleSettings language="JAVA">
    <option name="KEEP_CONTROL_STATEMENT_IN_ONE_LINE"
value="true"/>
    <option name="KEEP_BLANK_LINES_IN_CODE" value="1"/>
```

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

    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_PARAMETERS" value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_RESOURCES" value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_BINARY_OPERATION" value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_TERNARY_OPERATION"
value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_CHAINED_METHODS" value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_GROUP_FIELD_DECLARATIONS" value="false"/>
    <!-- Debatable -->
    <option name="ALIGN_CONSECUTIVE_VARIABLE_DECLARATIONS"
value="false"/>
    <!-- Debatable -->
    <option name="THROWS_LIST_WRAP" value="0" />
    <!-- Debatable -->
    <option name="ALIGN_MULTILINE_THROWS_LIST" value="false" />

    <option name="ALIGN_MULTILINE_FOR" value="false"/>
    <option name="CALL_PARAMETERS_WRAP" value="1"/>
    <option name="METHOD_PARAMETERS_WRAP" value="1"/>
    <option name="EXTENDS_LIST_WRAP" value="1"/>
    <option name="THROWS_KEYWORD_WRAP" value="1"/>
    <option name="METHOD_CALL_CHAIN_WRAP" value="1"/>
    <option name="BINARY_OPERATION_WRAP" value="1"/>
    <option name="BINARY_OPERATION_SIGN_ON_NEXT_LINE"
value="true"/>
    <option name="TERNARY_OPERATION_WRAP" value="1"/>
    <option name="TERNARY_OPERATION_SIGNS_ON_NEXT_LINE"
value="true"/>
    <option name="SPACE_BEFORE_METHOD_CALL_PARENTHESES"
value="false"/>
    <option name="FOR_STATEMENT_WRAP" value="1"/>
    <option name="ARRAY_INITIALIZER_WRAP" value="1"/>
    <option name="WRAP_COMMENTS" value="true"/>
    <option name="IF_BRACE_FORCE" value="3"/>
    <option name="DOWHILE_BRACE_FORCE" value="3"/>
    <option name="WHILE_BRACE_FORCE" value="3"/>
    <option name="FOR_BRACE_FORCE" value="3"/>
```

```
    <option name="PARENT_SETTINGS_INSTALLED" value="true"/>
    <indentOptions>
        <option name="INDENT_SIZE" value="4"/>
        <option name="CONTINUATION_INDENT_SIZE" value="4"/>
        <option name="TAB_SIZE" value="4"/>
        <!-- Debatable -->
        <option name="USE_RELATIVE_INDENTS" value="false"/>
    </indentOptions>
</codeStyleSettings>
```

3. Debatable

3.1. 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.

3.2. 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)) {

}
```

3.3. 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;
```

3.4. 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;
```

3.5. 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();
```

3.6. 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;
```

3.7. 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)

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

3.8. 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

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

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

Wrap if Long (value=1)

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

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

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")
```

3.9. 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) {
    }
```

3.10. 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);
```

3.11. 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 {
```

3.12. 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.

2.2. Explanation (Can skip)

2.2.1. 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");
}
```

2.2.2. 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);
```

2.2.3. 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) {
```

2.2.4. 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) {
}
```

2.2.5. 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 {
```

2.2.6. THROWS_KEYWORD_WRAP

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 {
```

2.2.7. 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
![](./imgs-code_style_guide/Team%20Code%20Style%20Configuration-1753679360584.png|516x31]]

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

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

2.2.8. 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

```
    int x = (3 + 4 + 5 + 6) * (7 + 8 + 9 + 10) * (11 + 12 + 13 + 14 +
```

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

```
int x =
    (3 + 4 + 5 + 6) * (7 + 8
        + 9 + 10) * (11 + 12
        + 13 + 14
        + 0xFFFFFFFF);
```

2.2.9. 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

```
int x = (3 + 4 + 5 + 6) *
    (7 + 8 + 9 + 10) *
    (11 + 12 + 13 + 14 +
        0xFFFFFFFF);
```

(Currently in Use) value=true

```
int x =
    (3 + 4 + 5 + 6) * (7 + 8
        + 9 + 10) * (11 + 12
        + 13 + 14
        + 0xFFFFFFFF);
```

2.2.10. 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

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

(Currently in use) value=true

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

2.2.11. 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

```
for (int i = 0;
        i < 0xFFFFFF; i += 2) {
    System.out.println (i);
}
```

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

```
for (int i = 0; i < 0xFFFFFF;
        i += 2) {
    System.out.println (i);
}
```

2.2.12. 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

```
int[] a = new int[]{1, 2, 0x0052, 0x0053, 0x0054};
```

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

```
int[] a = new int[]{1, 2,
        0x0052, 0x0053, 0x0054};
```

value=2, Chop down if long

```
int[] a = new int[]{1,
        2,
        0x0052,
        0x0053,
        0x0054};
```

2.2.13. 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)