

Code Style Guide

- [DSpace Java Style Guide \(Adopted 2018, for DSpace 7.x and above\)](#)
 - [Resources](#)
 - [IDE Support](#)
 - [IntelliJ IDEA](#)
 - [Eclipse](#)
 - [NetBeans](#)
 - [VSCode](#)
 - [Checking code style from commandline](#)
 - [Fixing existing code / PRs](#)
 - [Use IntelliJ to perform bulk cleanup](#)
 - [Old DSpace Java Style Guide \(for versions 6.x and prior\)](#)
- [DSpace Angular/Typescript Style Guide \(for DSpace 7.x and above\)](#)

DSpace Java Style Guide (Adopted 2018, for DSpace 7.x and above)

DSpace Java Style Guide is enforced on "main" branch

 As of Feb 2018, the below DSpace Java Style Guide is enforced on all Pull Requests to the "main" branch. Therefore, if a Pull Request to the "main" branch does not align with the below Style Guide, it will fail the build process within our GitHub CI.

- The enforcement is handled by [Checkstyle](#) (maven-checkstyle-plugin) using the Checkstyle configuration file in the root source directory: <https://github.com/DSpace/DSpace/blob/main/checkstyle.xml>
- If you would like to pre-check your Pull Request for any Code Style issues, you can do so by running: `mvn checkstyle:check`

1. 4-space indents for Java, and 2-space indents for XML. NO TABS ALLOWED.
2. **K&R style braces required. Braces are required on all blocks.**

```
if (code) {  
    // code  
} else {  
    // code  
}
```

3. Maximum length of lines is 120 characters (except for long URLs, packages or imports)
4. No trailing spaces allowed (except in comments)
5. Do not use wildcard imports (e.g. `import java.util.*`). Duplicated or unused imports are not allowed.
6. Write Javadocs for public methods and classes. Keep it short and to the point.
 - a. Javadoc `@author` tags are optional, but should refer to an individual's name or handle (e.g. GitHub username) when included
7. Tokens should be surrounded by whitespace, e.g. http://checkstyle.sourceforge.net/config_whitespace.html#WhitespaceAround

```
// This is NOT valid. Whitespace around tokens is missing  
String []={"one","two","three"}  
  
// This is valid. Whitespace exists around all tokens  
String [] = { "one", "two", "three" }
```

8. Each line of code can only include one statement. This also means each variable declaration must be on its own line, e.g.

```
// This is NOT valid. Three variables are declared on one line  
String first = "", second = "", third = "";  
  
// This is valid. Each statement is on its own line  
String first = "";  
String second = "";  
String third = "";
```

9. No empty "catch" blocks in try/catch. A "catch" block must minimally include a comment as to why the catch is empty, e.g.

```
try {  
    // some code ..  
} catch (Exception e) {  
    // ignore, this exception is not important  
}
```

10. All "switch" statements must include a "default" clause. Also, each clause in a switch must include a "break", "return", "throw" or "continue" (no fall throughs allowed), e.g.

```
// This is NOT valid. Switch doesn't include a "default" and is missing a "break" in first "case"
switch (myVal) {
    case "one":
        // do something
    case "two":
        // do something else
        break;
}

// This is valid. Switch has all necessary breaks and includes a "default" clause
switch (myVal) {
    case "one":
        // do something
        break;
    case "two":
        // do something else
        break;
    default:
        // do nothing
        break;
}
```

11. Any "utility" classes (a utility class is one that just includes static methods or variables) should have non-public (i.e. private or protected) constructors, e.g.

```
// This is an example class of static constants
public class Constants {
    public static final String DEFAULT_ENCODING = "UTF-8";
    public static final String ANOTHER_CONSTANT = "Some value";

    // As this is a utility class, it MUST have a constructor that is non-public.
    private Constants() { }
}
```

12. Each source file must contain the required DSpace license header, e.g.

```
/**
 * The contents of this file are subject to the license and copyright
 * detailed in the LICENSE and NOTICE files at the root of the source
 * tree and available online at
 *
 * http://www.dspace.org/license/
 */
```

Resources

IDE Support

Most major IDEs include plugins that support Checkstyle configurations. The plugin usually let you import an existing ["checkstyle.xml" configuration](#) to configure your IDE to use and/or validate against that style.

- [IntelliJ IDEA](#)
- [Eclipse](#)
- [NetBeans](#)
- [VSCode](#)

IntelliJ IDEA

(These instructions are based on IntelliJ 2021.2.3. They should apply to other recent versions, but some menus or settings may have changed.)

- Install the IntelliJ IDEA checkstyle plugin: <https://plugins.jetbrains.com/plugin/1065-checkstyle-idea>
 - File Settings Plugins
 - Restart IDE

- Configure it to use our custom checkstyle.xml for DSpace
 - File Settings Tools Checkstyle
 - In older versions of IntelliJ, this menu may be under File Settings Other Settings Checkstyle
 - Make sure to select a Checkstyle version which is *compatible with* the version we use in our Parent POM. Find the version in this tag in that POM:

```
<dependency>
  <groupId>com.puppycrawl.tools</groupId>
  <artifactId>checkstyle</artifactId>
  <version>8.30</version>
</dependency>
```

- Add a "Configuration File" (press the + in the table). Select our checkstyle.xml
- Make it "Active"
- Click Apply
- Update IDEA's Code Style to obey the Checkstyle Rules (See also <https://stackoverflow.com/a/35273850/3750035>)
 - File Settings Editor Code Style
 - Click the small gear icon next to "Scheme", Select "Import Scheme" CheckStyle Configuration
 - Select our checkstyle.xml
 - Click OK
 - Click Apply
- After importing the Checkstyle settings, your "Copyright profiles" may get reset (if you had them previously configured). If you want IntelliJ to automatically add a copyright statement / license at the top of every Java class, you should ensure this is (re)configured:
 - File Settings Editor Copyright
 - Add a Copyright profile. If you wish to use the DSpace copyright/license statement, the text of it is available in the [LICENSE_HEADER](#) file in the codebase (or see the Code Style Guide above)
 - Select the profile from the list, and adjust the formatting (unchecking the option "Add blank line after")
- Optionally, you also may wish to change some default IntelliJ settings, to allow the IDE to automatically reformat code for you (when making bulk edits). This is especially important if you plan to do any [bulk cleanup](#) of existing code using IntelliJ tools
 - First, fix the default order import statements per our CheckStyle rules (these don't seem to be auto updated by the Checkstyle plugin at this time)
 - File Settings Editor Code Style Java Imports
 - In the "Import Layout" section, ensure the settings are **in this order**
 - "import static all other imports"
 - <blank line>
 - "import java.*"
 - "import javax.*"
 - <blank line>
 - "import all other imports"
 - Once reordered, click OK
 - Then, update to auto-wrap lines at the right margin
 - File Settings Editor Code Style Java Wrapping and Braces
 - CHECK "Ensure right margin is not exceeded"
 - Click OK
 - Then, OPTIONALLY, for better looking bulk reformatting (i.e. results in more human-readable code), you may want to tweak which types of statements are set to "Wrap if long"
 - (NOTE: Without these settings, IDEA's bulk "Reformat Code" option sometimes will generate line breaks wherever the line ends, which can result in somewhat odd looking code. These settings cleanup some of the more odd line wrappings that may occur.)
 - File Settings Editor Code Style Java Wrapping and Braces
 - Optionally, **set all of the following** to "Wrap if long":
 - Extends/implements list (also check "align when multiline")
 - Extends/implements keyword
 - Throws list
 - Throws keyword
 - Method declaration parameters (also check "align when multiline")
 - Method call arguments (also check "align when multiline")
 - Chained method calls (also check "align when multiline")
 - "for()" statement (also check "align when multiline")
 - "try-with-resources" (also check "align when multiline")
 - Click OK
 - Finally, update IDEA's Javadoc settings to **not** insert "<p>" on blank Javadoc lines (as this seems to cause IDEA to change all our DSpace license headings improperly during bulk editing)
 - File Settings Editor Code Style Java Javadoc
 - UNCHECK "Generate <p> on empty lines"
 - Make sure "Keep empty lines" is checked
 - Click OK

Eclipse

These instructions were created using Eclipse Neon. Note: Eclipse Che (Codenvy) does not seem to support checkstyle.

- Use the Eclipse Checkstyle plugin: <https://checkstyle.org/eclipse-cs/#!/>
 - Note that you can drag the plugin into your Eclipse installation
- Right click your DSpace project; select Properties Checkstyle
- Go to "Checkstyle Local check configurations" and click "New"
 - Create a "Project Relative Configuration"

- Name: DSpace Internal
 - Location: checkstyle.xml
- An "Unresolved Properties Error" box will appear; click "Edit Properties"
 - Add a new property
 - Name: checkstyle.suppressions.file
 - Value: full path to checkstyle-suppressions.xml
 - Click OK
- Go to "Checkstyle Main"
 - Select "DSpace Internal" from the drop down
 - Check "Checkstyle active for this project"
 - Click OK
- Code will be rebuilt
- Edit a java file. Place a curly brace on a line by itself.
 - The line of code should get highlighted in red.

Tested and works to validate code. To format you must generate a formatter style following the [steps here](#). It seems impossible to force eclipse to use braces on single-line if statement

(Please help us enhance these instructions to provide a step-by-step configuration for Eclipse)

NetBeans

These instructions worked for NetBeans 8.2.

- Download the NetBeans Checkstyle plugin: <http://plugins.netbeans.org/plugin/3413/checkstyle-beans>
 - Go to "Tools Plugins Downloaded Add Plugins...":
 - Add the downloaded plugin and install it! You will have to restart NetBeans.
- Go to "Tools Options Miscellaneous Checkstyle":
 - Add the Path of our [checkstyle.xml](#) as "Configuration file" and put the "Properties" as: checkstyle.suppressions.file=Path to our [checkstyle-suppressions.xml](#)
 - Click on Apply than on OK.

The plugin still works in Apache NetBeans 12.6. It can be obtained at <https://www.sickboy.cz/checkstyle/download.html>. The instructions above should still work.

VSCode

These instructions are for VSCode version 1.63.2.

- Open the Extensions panel. Search for and install the Checkstyle for Java extension by ShengChen. Follow the [configuration docs for the plugin](#). The important parts are: set the configuration file, by right clicking the checkstyle.xml file and select Set the Checkstyle Configuration File. You will need to use the command Checkstyle: Set the Checkstyle Version (Help>Show All Commands, or hit F1 on your keyboard) to manually set the Checkstyle version to 8.30. The extension will automatically download the required checkstyle jar file if it needs to. Other versions of Checkstyle jar are known to not work (8.18 and 9.2.1 as of this writing all currently throw errors). The DSpace pom.xml file [pins the version of checkstyle to 8.30](#), we should use that version in VSCode.
- You'll need to restart VSCode after adding the Checkstyle for Java extension
- You will want to modify the setting for Java Completion: Import Order (in the settings.json file for VSCode): add this to your settings.json file:


```
"java.completion.importOrder": ["#", "java", "javax", "com", "org"],
```
- Whenever you select an option to "organize imports", VSCode will incorrectly add a single blank line in your imports block, where our checkstyle configuration says it should not. The checkstyle plugin will alert you to this fact, and you can remove it manually. Unfortunately, VSCode does not currently support specifying the exact importOrder in the java.completion.importOrder that we require. But this is close enough.

Checking code style from commandline

If you do not use one of the above IDEs to write your code, you can always verify your code from the command-line using the following Maven command. This command can be run from the root source directory ([dspace-source]) to check all source code, or from within an individual module (e.g. [dspace-source]/dspace-api/) to only check that module.

```
# If the below checkstyle:check command complains about missing SNAPSHOT dependencies, then you may need to
first run:
# mvn install

# This checks the code style of all source code under the current directory
# It also ensures all source code has the required license header.
mvn -U checkstyle:check license:check
```

Fixing existing code / PRs

Use IntelliJ to perform bulk cleanup

- First, you MUST follow **all the configurations** in the [IntelliJ configuration](#) section above. Especially pay attention to the settings marked optional, unless you are doing bulk edits. Since you want to do bulk edits, those settings are now required.
- Now open up any Java source file. You'll see Checkstyle warnings appear (if any) in RED.
 - This highlighting is enabled via the checkstyle plugin.
- **Bulk Reformat (an entire module):** You can also reformat code in bulk in IDEA: https://www.jetbrains.com/help/idea/editor-basics.html#reformat_rearrange_code
 - Right click on a single directory / module
 - Select "Reformat Code"
 - Check "Optimize imports" (This will remove any import statements with an asterisk)
 - Leave "Rearrange entries" unchecked (This will rearrange methods/variables based on arrangement settings in IDEA. We don't need this)
 - Under "Filters" Check "File Masks" and enter in "*.java, *.xml" (As we'll only bulk reformat Java and XML files)
 - Click Run
- Finally, run Checkstyle against the module. There likely will still be a few failures to fix manually.

```
# mvn install (Only necessary if checkstyle:check below complains about missing SNAPSHOT dependencies)
mvn -U checkstyle:check
```

- Also double check license headers are still accurate. Sometimes these get "munged" during bulk reformat (as IDEA occasionally tries to merge nearby comments into one larger comment)

```
mvn license:check
```

Old DSpace Java Style Guide (for versions 6.x and prior)

Per the [Code Contribution Guidelines](#) page (see "Coding Conventions" section), our existing style guide is listed as follows:

Your code needs to follow the [Sun Java code conventions](#) with the following minor modifications:

- *Curly braces must be on new lines.*
- *Source files must have a copy of the copyright Duraspace notice and BSD license at the top (see [Licensing of Contributions](#) below). Also take a look at [Copyright and Licensing](#).*
- *You must use 4-space tabulation.*
- *'else' should be on a new line. 'else if' stays on one line.*
- *Users of the Eclipse IDE can have eclipse do the formatting automatically using this profile: - [Dspace-eclipse-format.xml](#). See the Eclipse section below for details of how to apply this profile.*

Your code should be well commented with Javadoc (including package and class overviews). All code contributions must come with Documentation. At a bare minimum, this should include Technical Documentation covering all configuration options and/or setup. See [Documentation Contributions](#) below for more details.

These older style guidelines were based heavily on the Sun coding conventions (circa 1999) which are no longer maintained, but still available at <http://www.w.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>

Because the Sun Java style guide is no longer maintained, it will not be keeping up with current Java style best practices, features, etc.

In 2018, we decided to update our guide based on / inspired by these two guides:

- The Fedora project style guide at [Code Style Guide](#)
- The Google Java Style Guide <https://google.github.io/styleguide/javaguide.html>

DSpace Angular/Typescript Style Guide (for DSpace 7.x and above)

DSpace Typescript Style Guide is enforced on "main" branch

 The DSpace Typescript Style Guide is enforced on all Pull Requests to the "main" branch. Therefore, if a Pull Request to the "main" branch does not align with the below Style Guide, it will fail the build process within our GitHub CI.

- The enforcement is handled by [ESLint](#) using the configuration file in the root source directory: <https://github.com/DSpace/dspace-angular/blob/main/.eslintrc.json>
- If you would like to pre-check your Pull Request for any Code Style issues, you can do so by running: `yarn lint`

For the DSpace Angular UI (written in TypeScript), we use [ESLint](#) to validate the style of all Typescript (*.ts) files.

- All style rules are defined in <https://github.com/DSpace/dspace-angular/blob/main/.eslintrc.json>
- Most IDEs include a ESLint plugin which can automatically enforce these style rules in your Typescript code.