APPENDIX A

Customizing IntelliJ

What we'll cover:

- Code Style
- Colors and Fonts
- Keyboard shortcuts

IntelliJ is a very capable IDE and very customizable too. In this chapter, we'll look at the various ways you can customize your environment exactly to your liking.

Code Style

If you're a professionally employed programmer, chances are your company follows a coding standard or guideline. IntelliJ makes it easy to follow those guidelines.

The settings for coding style are found on the *Preferences* (macOS) or *Settings* (Windows and Linux) — press **`cmd + ,`** for macOS or **`CTRL + ALT + S`** for Windows/Linux. Figure A-1 shows the Preferences dialog.

T. Hagos, *Beginning IntelliJ IDEA*, https://doi.org/10.1007/978-1-4842-7446-0

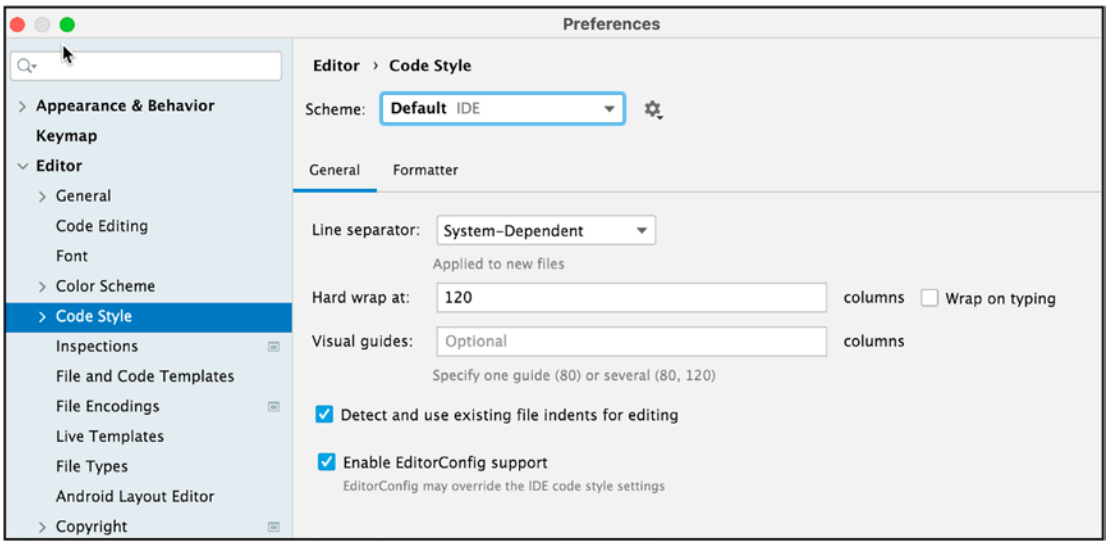
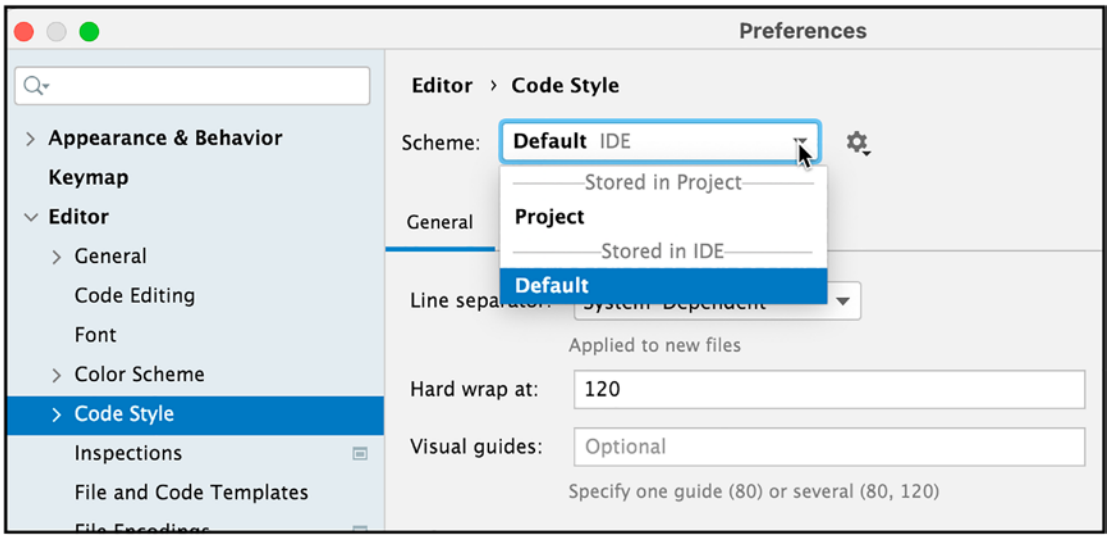


Figure A-1. *Preferences*

I'm sure you're already familiar with this dialog because we referred to this dialog many times in the previous chapters, but let's take a look at the **Editor ➤ Code Style** section in particular.

Code styles are defined at two levels, at a project level, and at the IDE (global) level. When you first open the **Code Style** settings, the scheme is set to IDE (as you can see in Figure A-1, above). This means that whatever changes you make to the Code Style settings, you're making at a global level. It will affect all currently opened projects and other projects you'll create in the future. If you don't want to change the settings for all projects, you should instead switch the scheme to "Project"; to do that, click the down-arrow of the *Scheme* dropdown (as shown in Figure A-2) then choose *Project*.



this figure will be printed in b/w

Figure A-2. *Switch scheme*

Language settings

IntelliJ supports quite a number of programming languages (even the Community Edition). You can change a lot of the IDE's behavior for a specific language. Drill down on the Code Style menu (as shown in Figure A-3), and choose a language to customize.

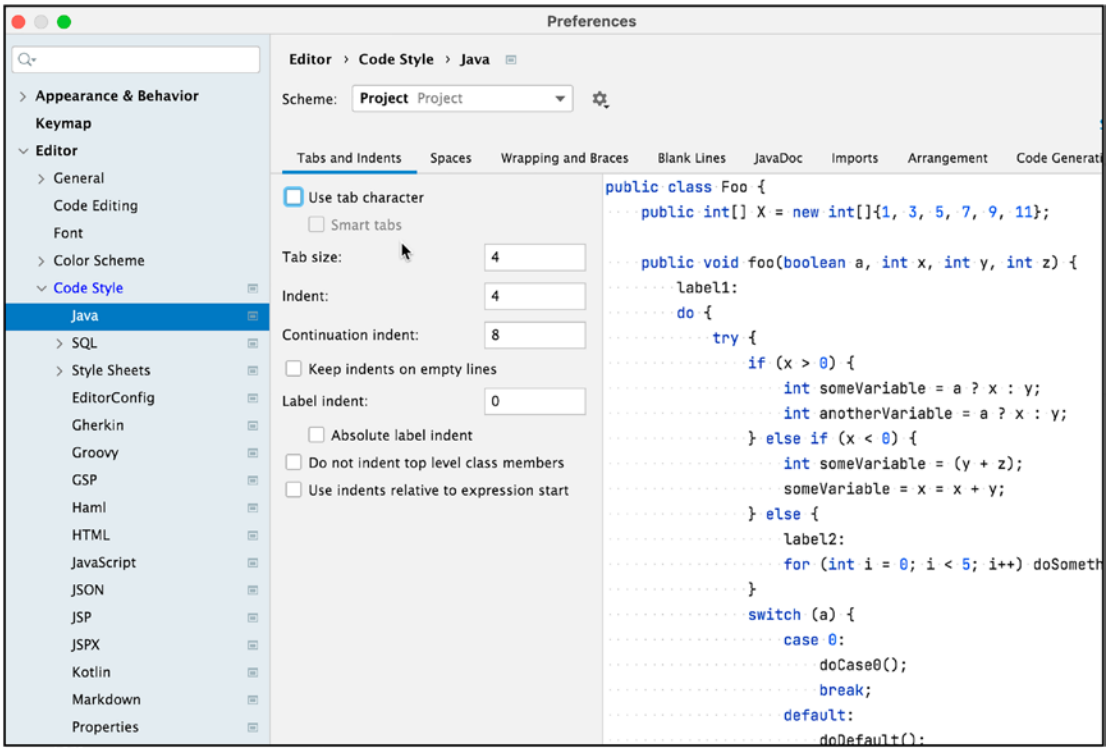
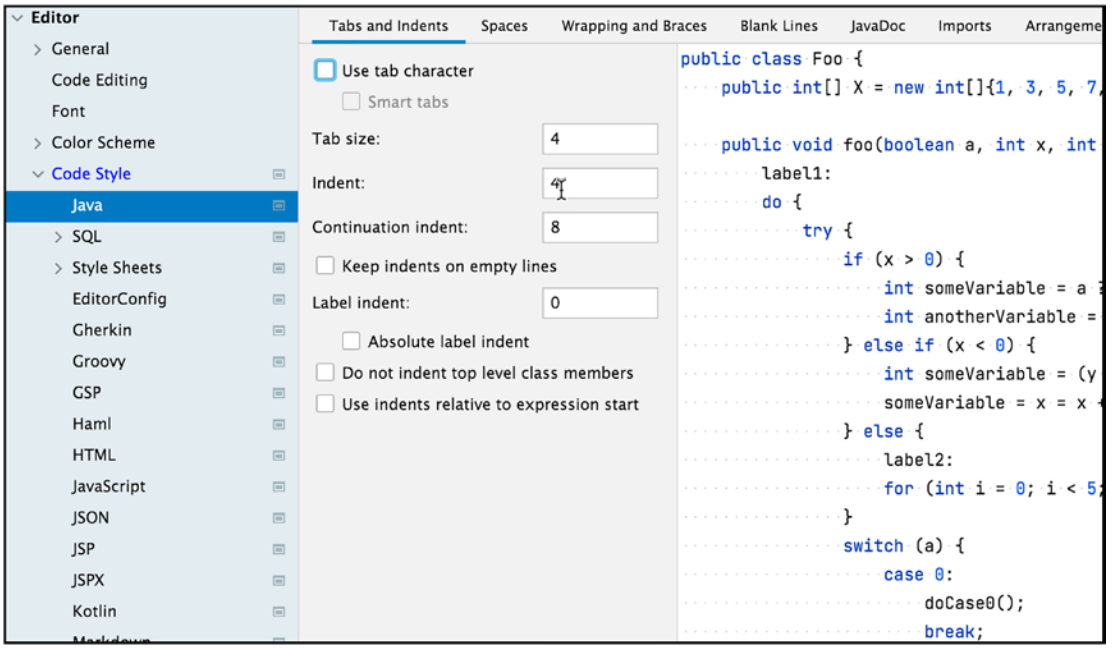


Figure A-3. *Java language settings*

In Figure A-3, I chose to customize the Java language. There are quite a few things you can change here.

Tab and Indents

If your coding standards call for the use of spaces instead of the tab character, this is the place to change it. By default, IntelliJ uses spaces instead of tabs; and it uses four spaces for one tab position. You can change this, of course. Figure A-4 shows the dialog for Tab and Indents.



this figure will be printed in b/w

Figure A-4. *Tab and Indents*

As you make changes to the settings, IntelliJ shows the effect of those changes on a preview panel, as you can see in Figure A-4.

Spaces

If you want to remove the whitespace between some control structures and the parentheses, for example, this is where to do it. Figure A-5 shows the Spaces dialog.

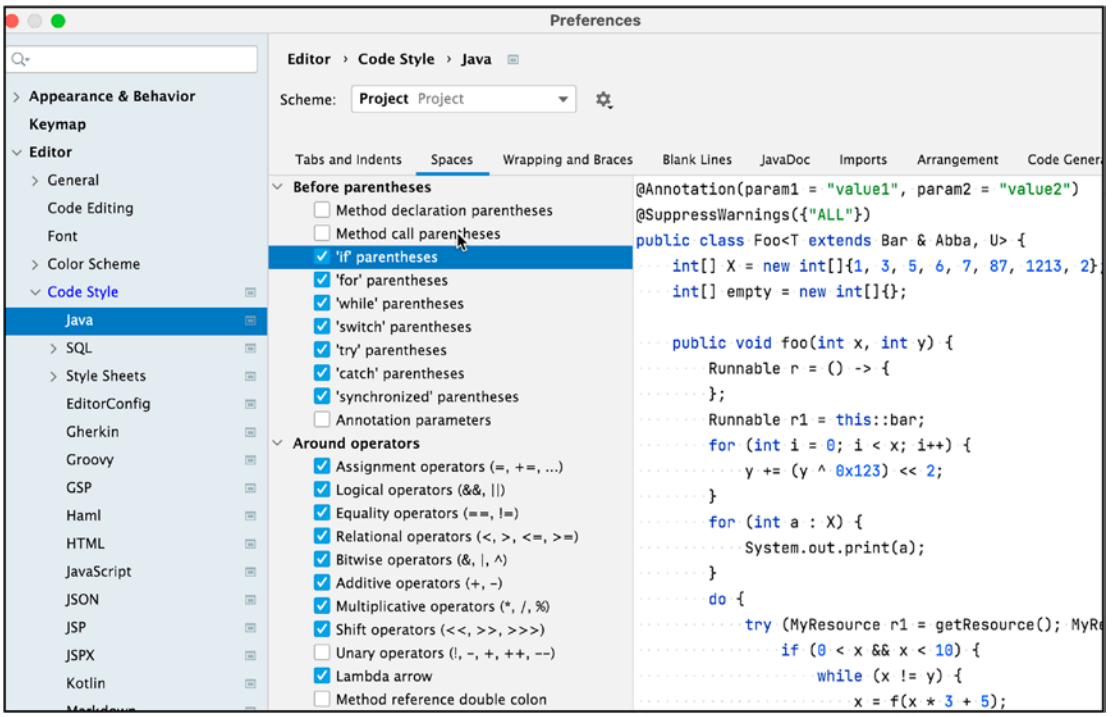


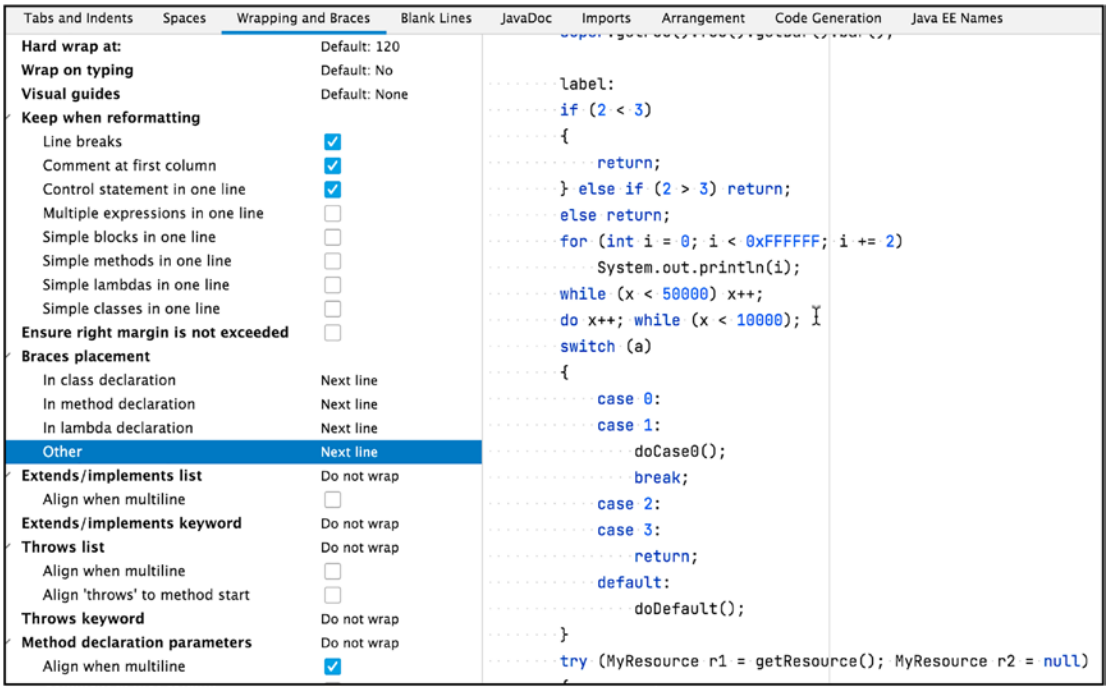
Figure A-5. *Spaces dialog*

Wrapping and Braces

By default, IntelliJ follows the K & R (Kernighan and Ritchie) convention for braces; that is, your code will look like this

```
public class MainProgram {
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

The opening curly brace is at the end of the line instead of on a new line. If you need to follow the C++ or C# convention where the opening curly brace is on a new line, then change the settings of the Braces placement to “new line” instead of “end of line”, as shown in Figure A-6



this figure will be printed in b/w

Figure A-6. *Wrapping and Braces*

I'll leave the rest of the sections for you to explore, but I'd like to look at the **Code Generation** settings again — I say again because we've already discussed this in Chapter 5 (Navigation and Generation).

In Figure A-7, I placed an "m" in the *Name prefix* section of the **Field,** and an "s" in the *Name prefix* of **Static Field**. I'm telling IntelliJ how I am prefixing my member variables. This is useful when you're generating getter and setters. IntelliJ will know which characters to ignore when it generates the getter and setter methods.

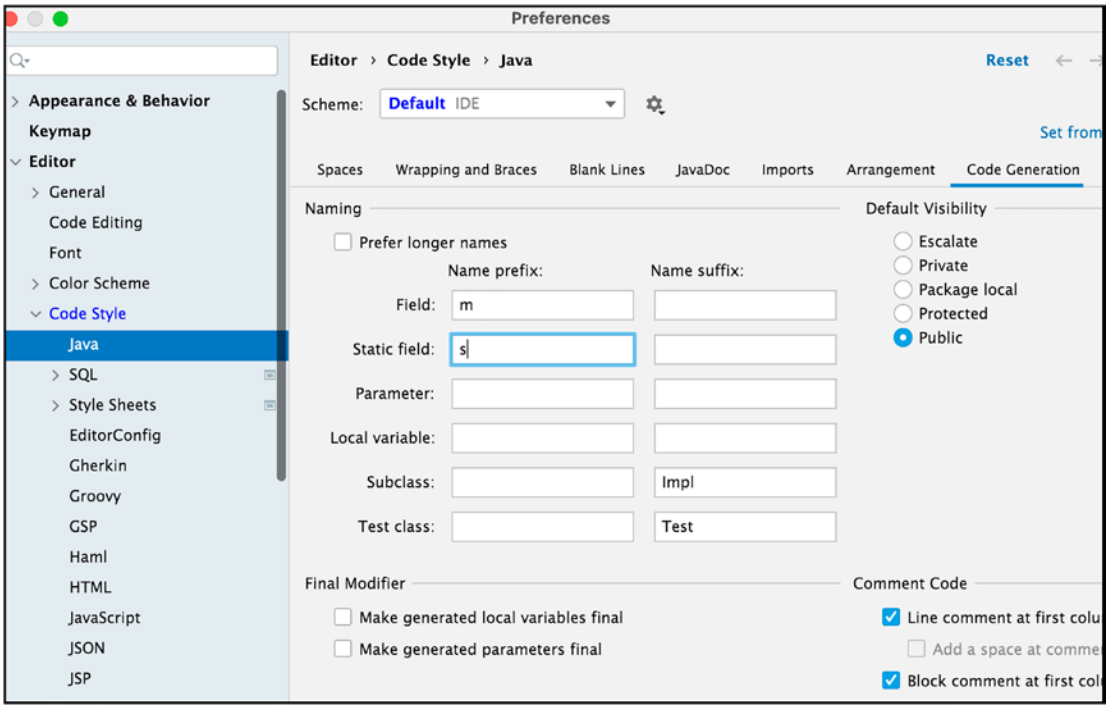
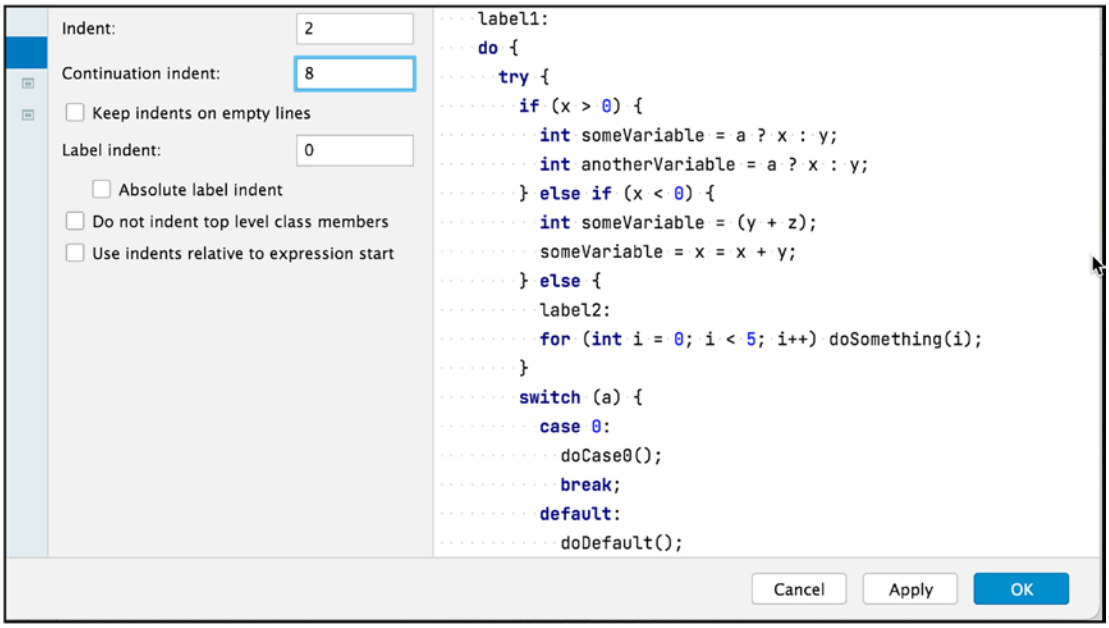


Figure A-7. *Code Generation*

When you're happy with your changes, don't forget to click the *Apply* button (Figure A-8) to save your modifications. Clicking Apply won't leave the Preferences dialog; it will simply save your changes. If you click the OK button, that will both save your changes and exit the Preferences dialog.



this figure will be printed in b/w

Figure A-8. *Apply button*

If you work using other programming languages and you'd like to use the same settings as we've done in the Java language section, you can use the "Set from" feature of IntelliJ (shown in Figure A-9).

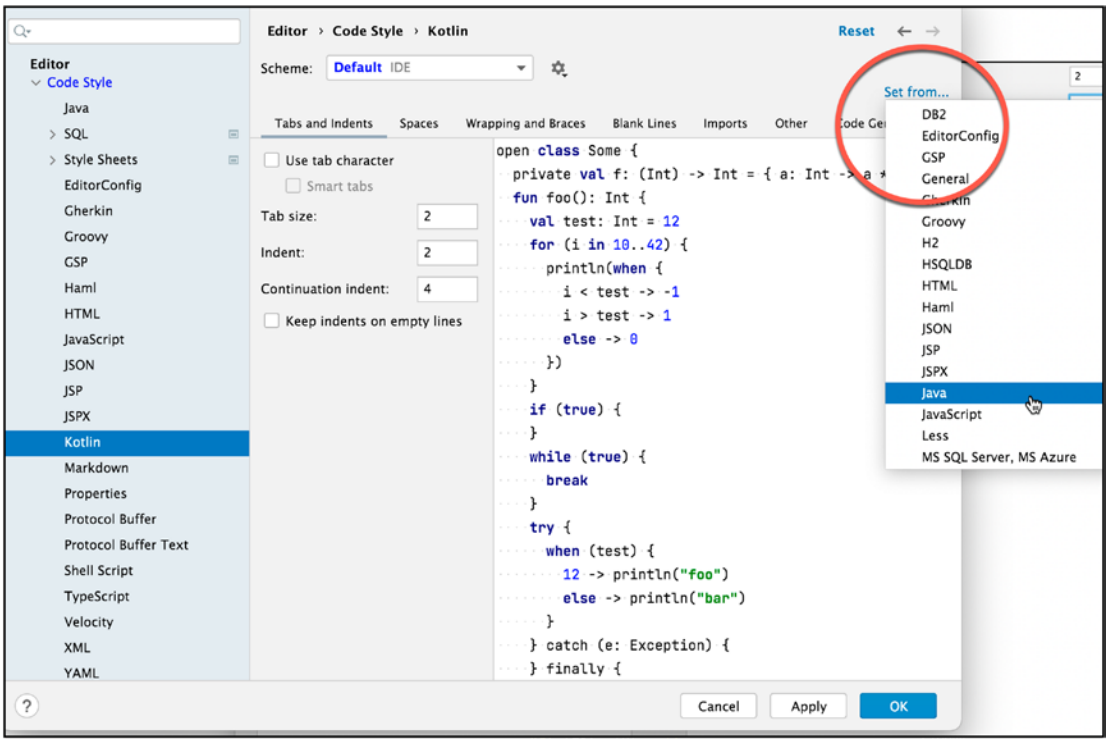


Figure A-9. *Set from*

As you can see, when I moved to the Kotlin language, I simply clicked the "Set from" link, then chose Java. All the language settings of Java are now inherited in my Kotlin settings. It's a faster way of configuring your language settings instead of doing it manually for each programming language you'd like to set.

Before we leave this section, I'd like to point out the "Reset" link (shown in Figure A-10).

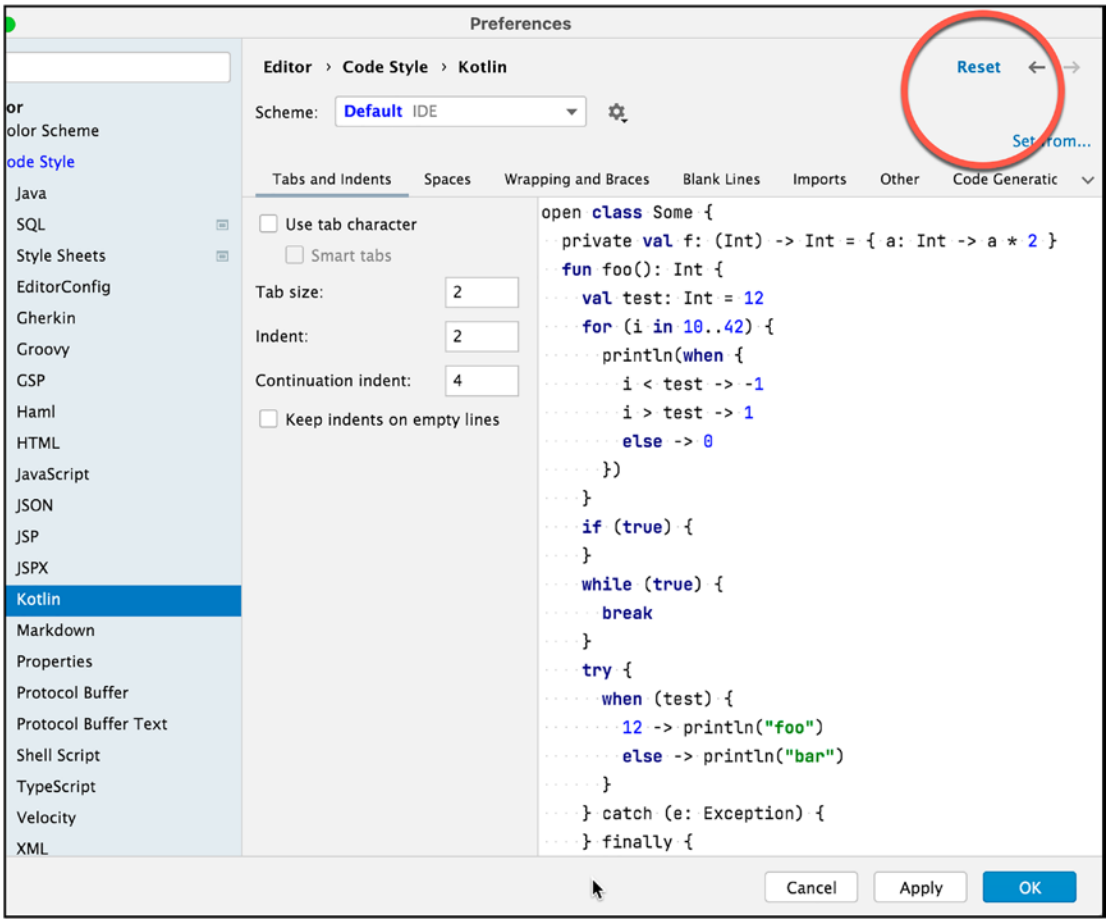


Figure A-10. *Reset*

As you might have guessed, the "Reset" will roll back all of your changes — since you last clicked the *Apply* button. But once you click the Apply button, you won't be able to roll back anymore.

Colors and Fonts

One of IntelliJ's biggest benefits to developers is syntax highlighting. By default, the IDE has already decided how to color (and style) the variables, keywords, comments, control structures, etc.; but if you're not happy with the defaults, the *Color Scheme* section of the Preferences dialog is where you can make changes. Clicking the *Color Scheme* section shows you the current color scheme of the editor (Figure A-11).

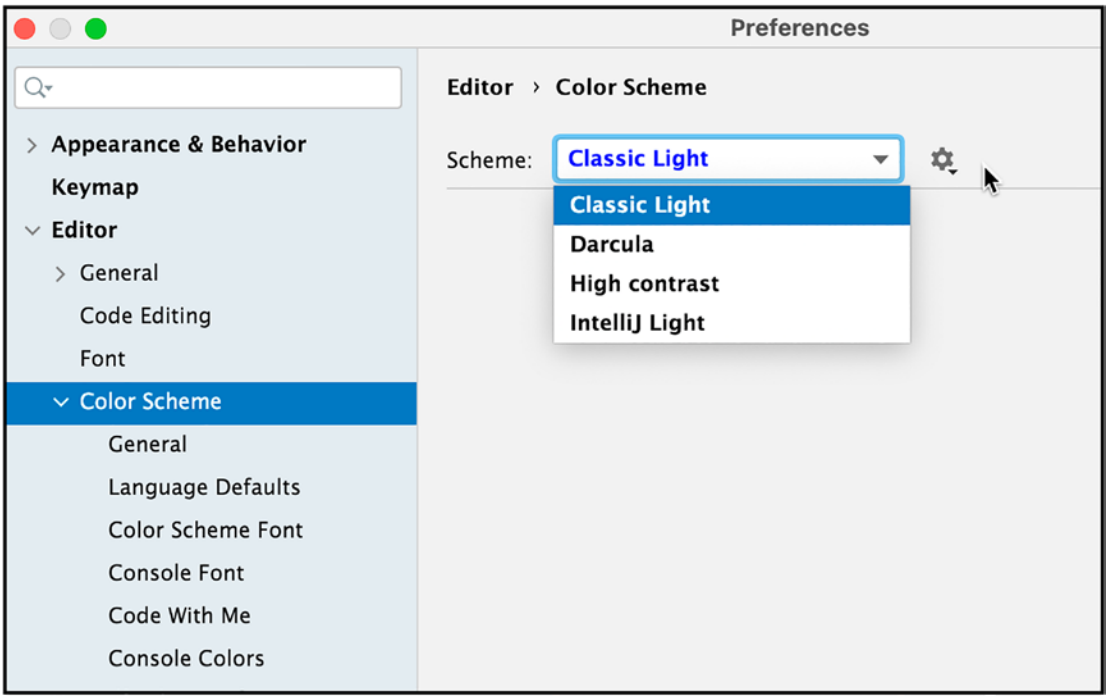


Figure A-11. *Color Scheme*

I bet you've already tried this out and have already settled on your color scheme. You can simply choose from the four color schemes of the IDE, but if you feel you need to tinker a bit more, you can customize it further. You cannot directly edit the existing schemes; you have to create a duplicate first, then edit that. Click the gear icon right next to the Scheme dropdown (as shown in Figure A-12), then choose "Duplicate".

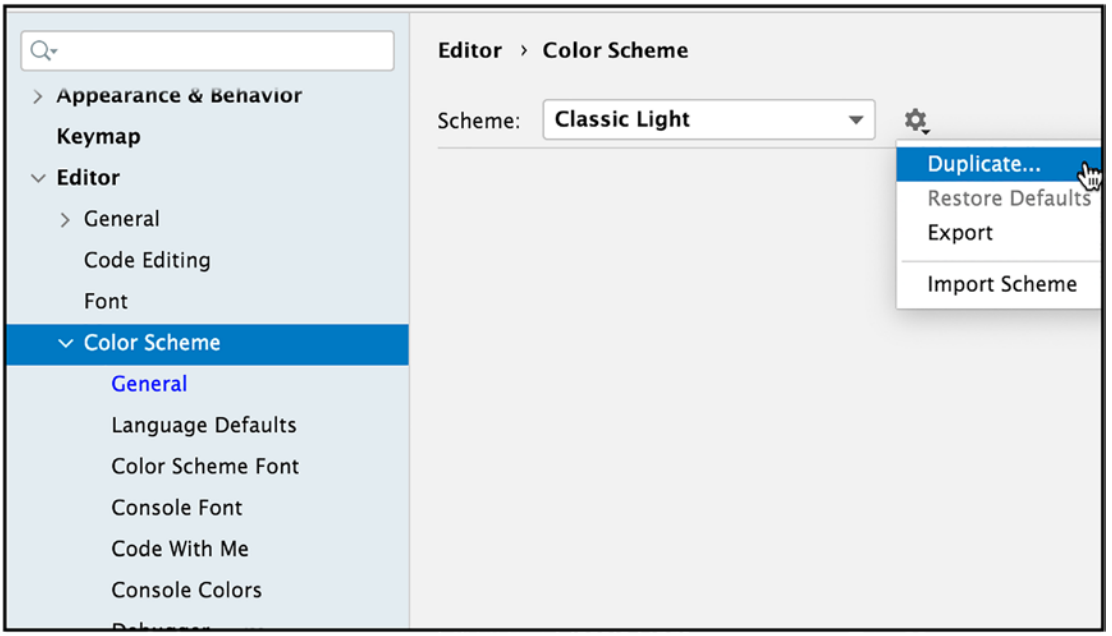


Figure A-12. *Duplicate color scheme*

As soon as you click "Duplicate", the Scheme dropdown turns to an edit field. Type the name of your modified scheme here (as shown in Figure A-13). Then press ENTER.

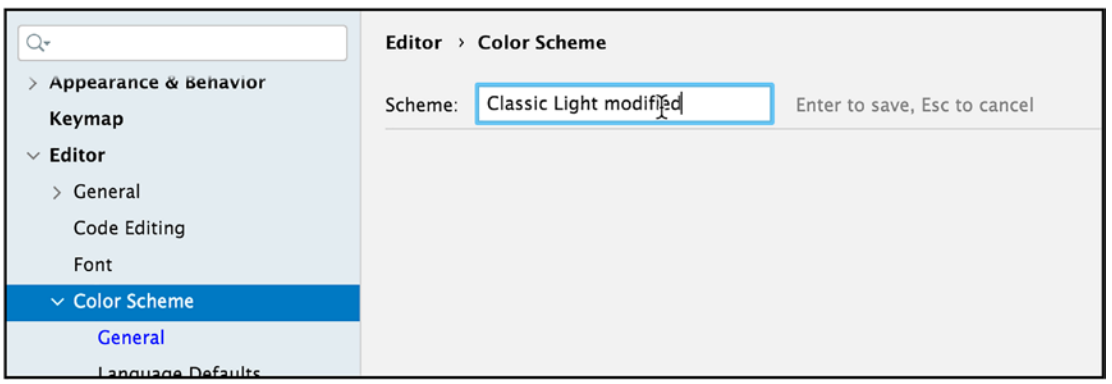


Figure A-13. *New Scheme name*

Now we can make some changes to the Color scheme.

The **General** section defines basic editor colors, like the gutter, line numbers, errors, warnings, popups, hints, etc. The **Language Defaults** section contains common syntax highlighting settings, which are applied to all supported programming languages by default. Generally, you only need to make changes to the Language Defaults (Figure A-14), then make adjustments for the specific languages (if you really need to). To change inherited color settings for an element, clear the **Inherit values from** checkbox.

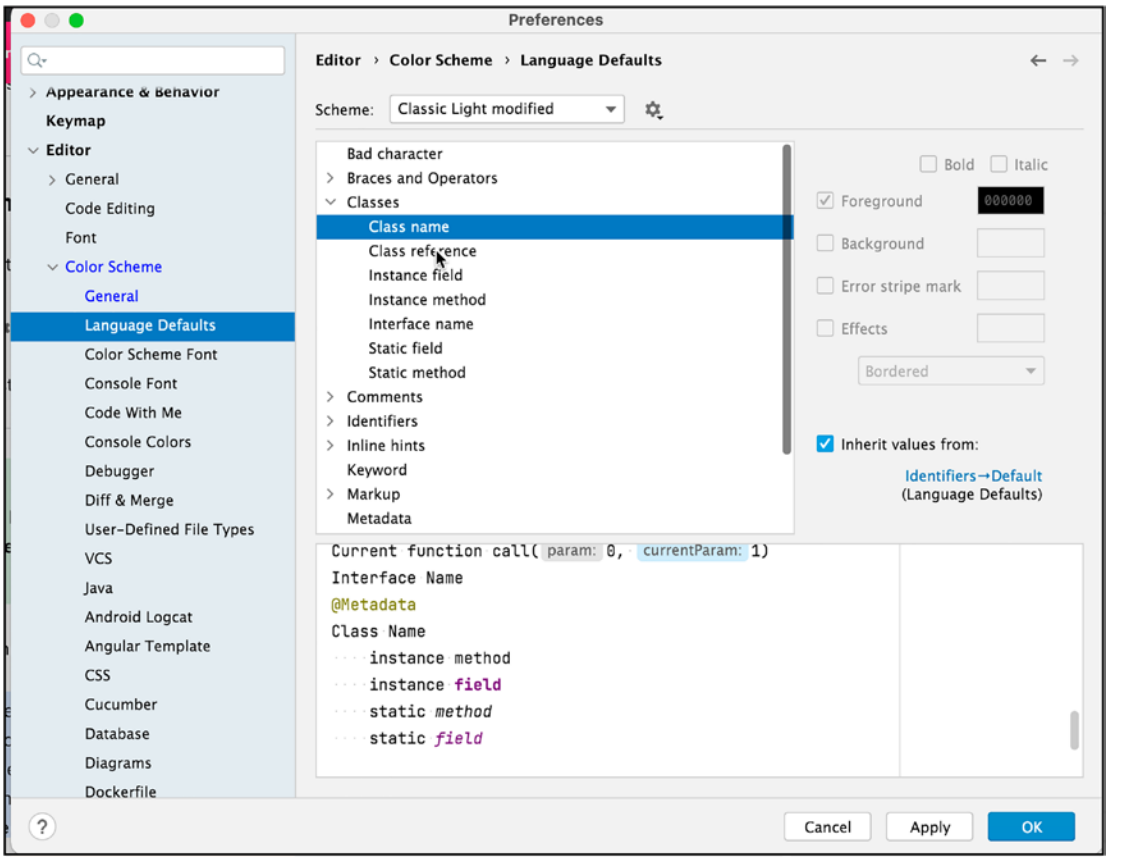


Figure A-14. *Language defaults*

Semantic highlighting

The color scheme defines syntax highlighting for reserved words and other symbols in your source code: operators, keywords, suggestions, string literals, and so on. If you have a function or method with many parameters and local variables, it may be hard to distinguish them from one another at a glance. You can use *semantic highlighting* to assign a different color to each parameter and local variable.

The settings for Semantic highlighting are under **Editor ➤ Color Scheme** of the Preferences dialog. Choose Language Defaults — as you can see in Figure A-15 — then choose Semantic highlighting. After that, enable the "Semantic highlighting" checkbox and customize the individual colors as you see fit.

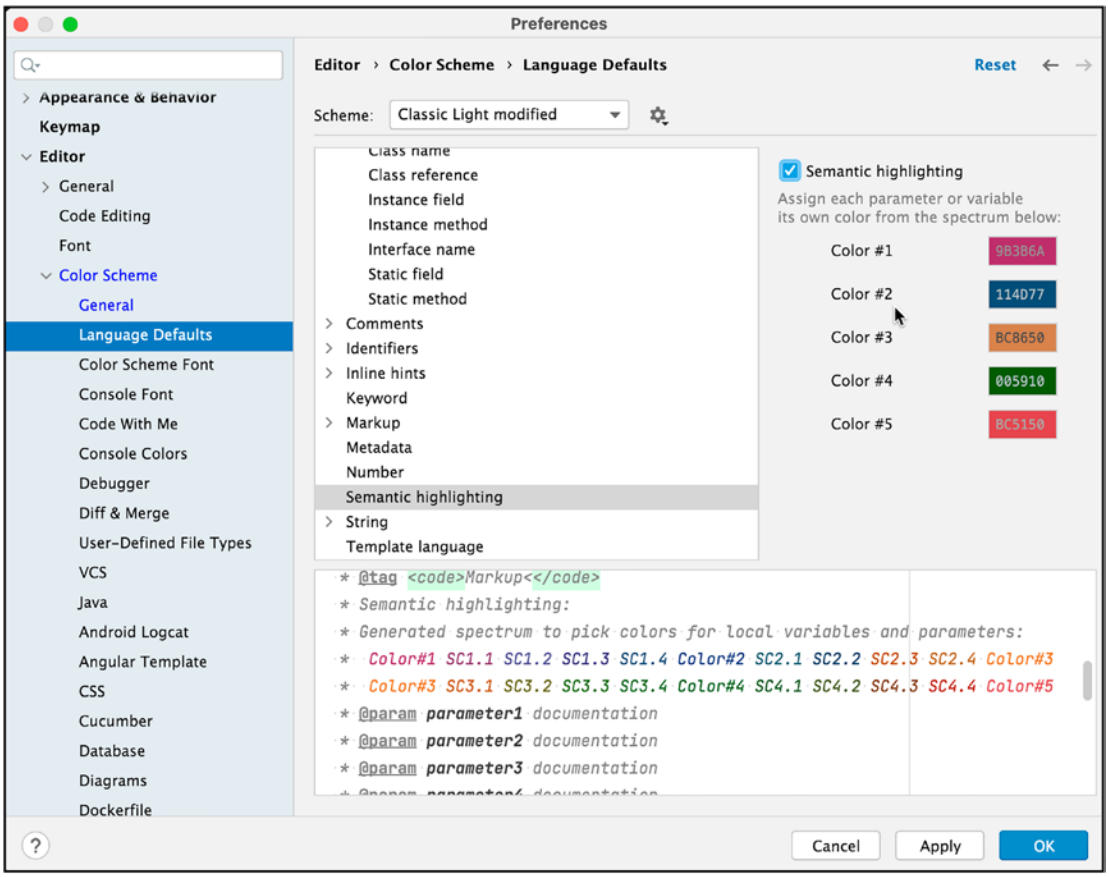


Figure A-15. *Semantic highlighting*

this figure will be printed in b/w

This will enable semantic highlighting for all languages that inherit this setting from **Language Defaults**. To enable it for a specific language instead (for example, Java), go to the **Editor | Color Scheme | Java | Semantic highlighting** page of the Preferences (or Settings) dialog, clear the **Inherit values from** checkbox, and select the **Semantic highlighting** checkbox.

Share color schemes

Customizing colors can be tedious, especially if you have to do it on multiple machines. Thankfully, you can export color schemes from one installation and then import them from another one. If you're coming from Eclipse, you can actually import your favorite color settings from Eclipse.

To export a color scheme, you need to save it as an XML file with the **.icls** extension. Then you can import it from another installation.

From **Editor ➤ Color Scheme** (shown in Figure A-16), click the gear icon, then choose **Export ➤ IntelliJ color scheme (.icls)**

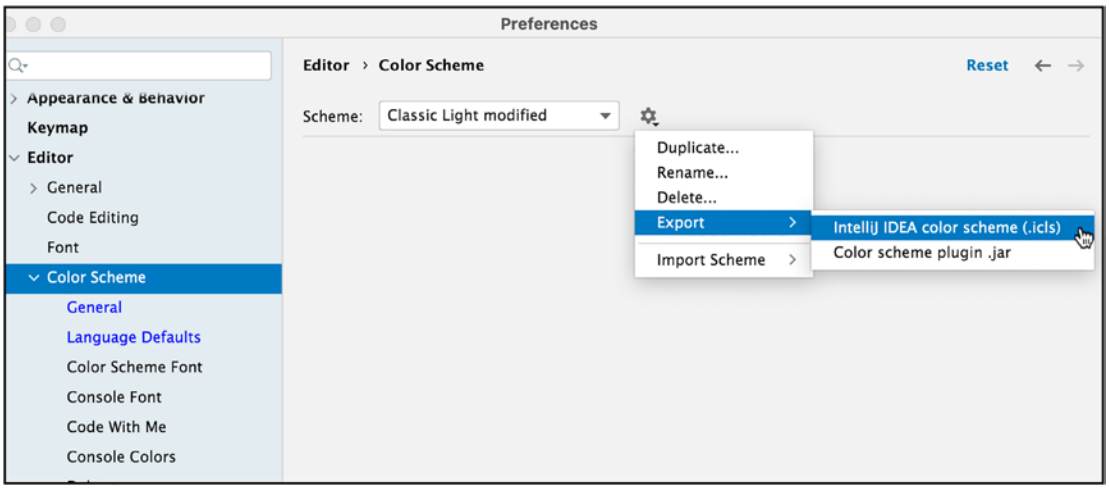


Figure A-16. *Export color scheme*

The dialog that follows will let you save the exported color scheme file (shown in Figure A-17),

this figure will be printed in b/w

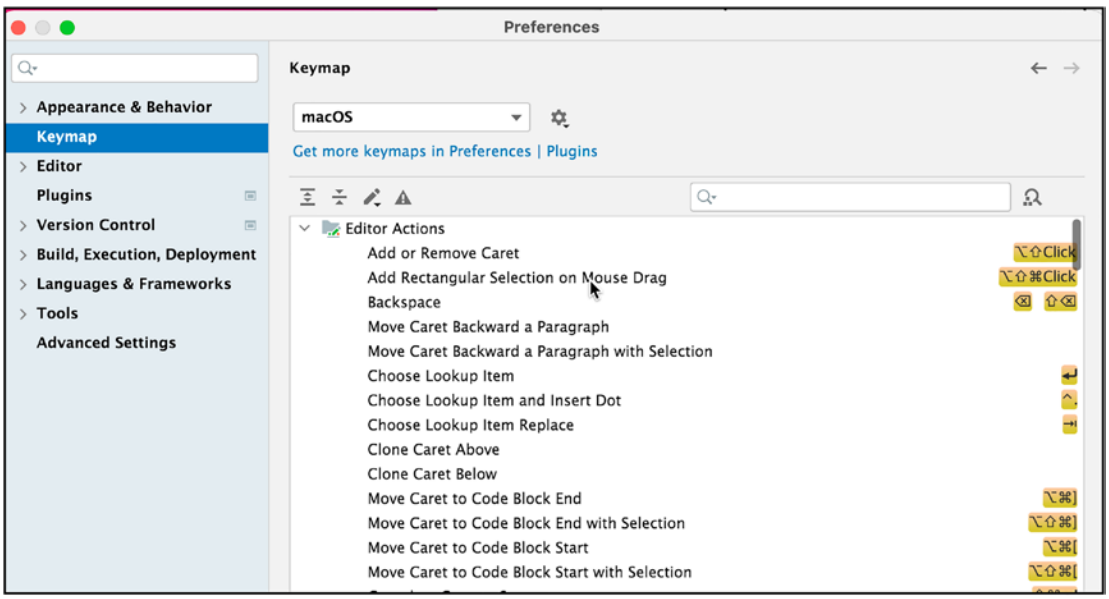
Figure A-17. *Target File*

You can then import this file from another installation. To import a color scheme file, use the same technique when we exported the color scheme — Editor ➤ Color Scheme ➤ Import Scheme

Keyboard shortcuts

Out of the box, IntelliJ already includes many predefined keymaps, but it also lets you customize these keyboard shortcuts.

To view the existing keymap, open the Preferences or Settings dialog, then go to Keymap (as shown in Figure A-18).



this figure will be printed in b/w

Figure A-18. *Keymaps*

By default, IntelliJ chooses the keymap that's more appropriate for your OS, but you can switch to another keymap by clicking the keymap dropdown (shown in Figure A-19

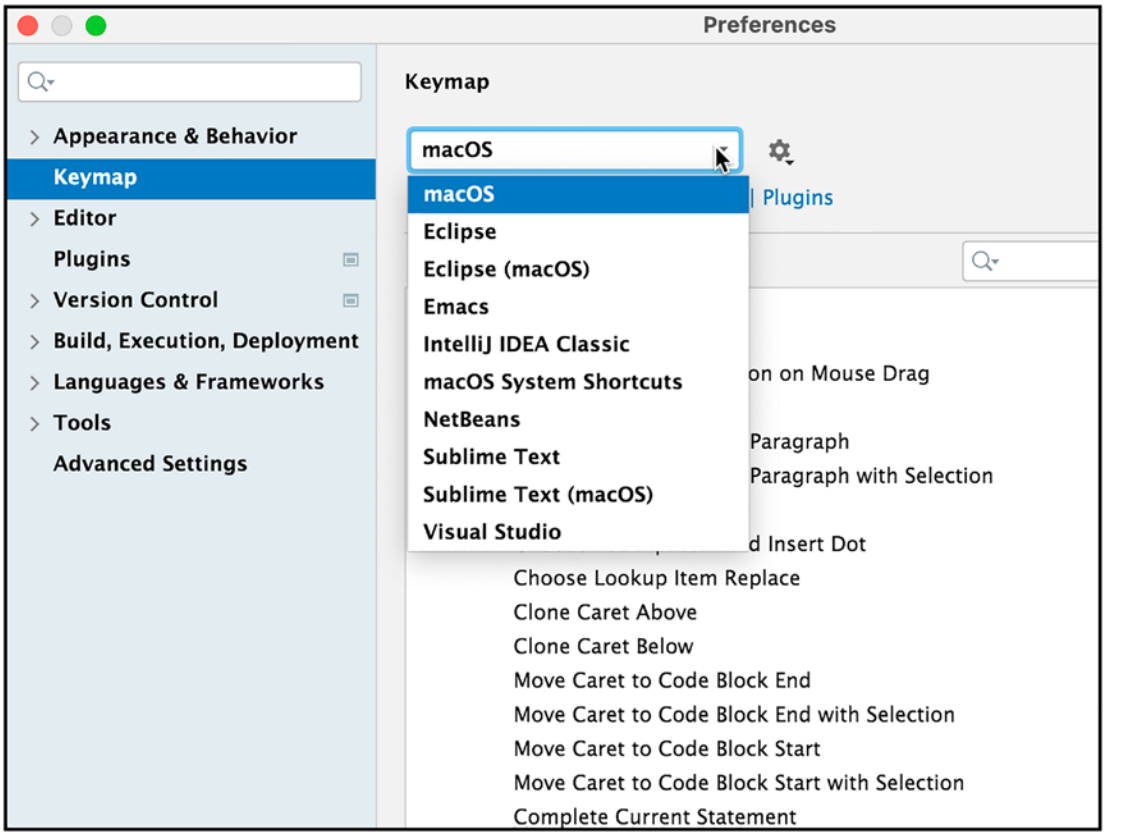


Figure A-19. *Switch keymaps*

Just like the Color Scheme, you cannot directly edit an existing keymap; you'll have to create a duplicate of an existing keymap, then customize that. To duplicate a keymap, click the gear icon right next to Keymap dropdown (as shown in Figure A-20).

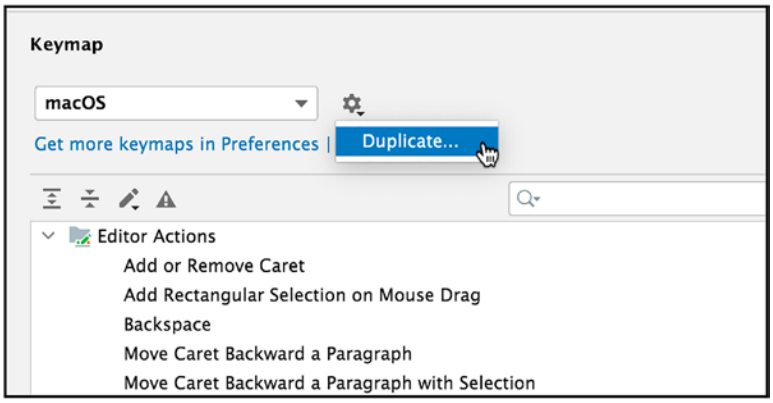


Figure A-20. *Duplicate an existing keymap*

Click "Duplicate," then name your new keymap. Now, you can make changes to the existing keyboard shortcuts.

To add a new keyboard shortcut, choose any action from the keymap, right-click, then "Add Keyboard Shortcut" — as shown in Figure A-21.

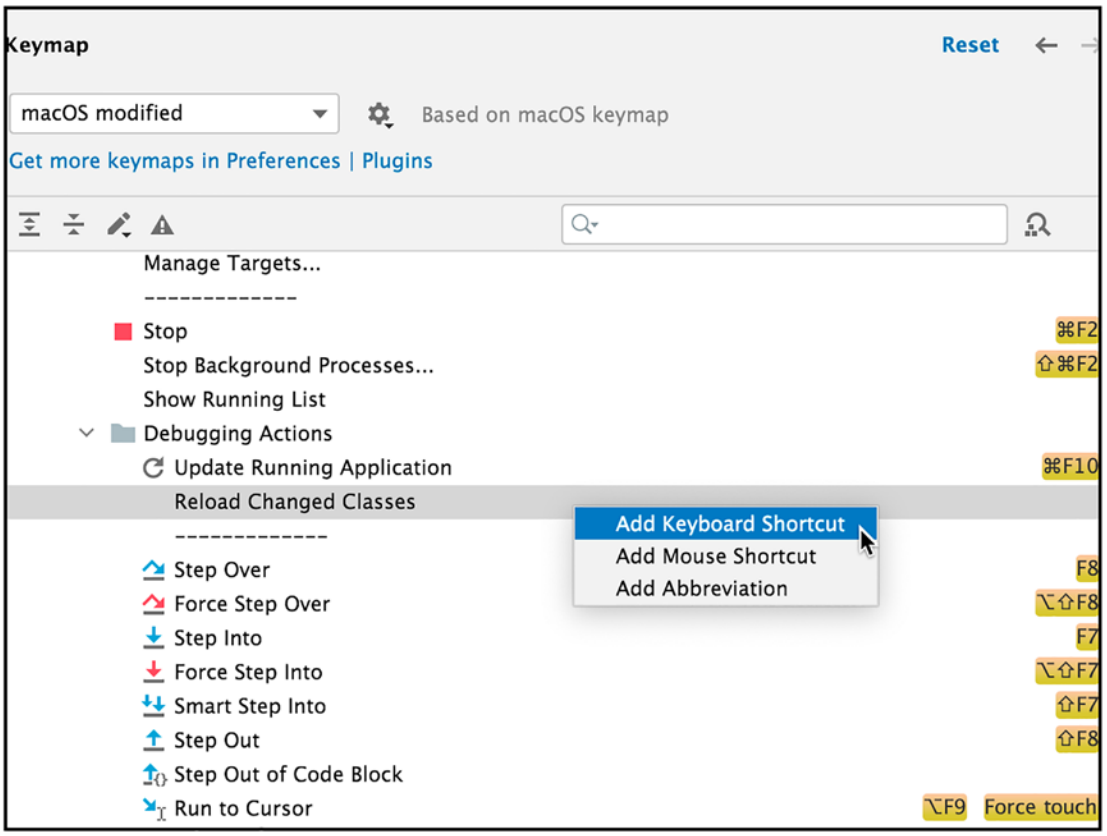


Figure A-21. *Add a Keyboard shortcut*

In the keyboard shortcut dialog, press your desired key combination, then click the OK button to save the shortcut.

Key Takeaways

- IntelliJ comes with lots of sensible defaults, but it also leaves room for customization
- Coding guidelines are a big deal for teams. IntelliJ makes it easy to follow any kind of coding style