

IntelliJ productivity hints (beyond keymap)

Urs Blum

IntelliJ IDEA
WINDOWS & LINUX KEYMAP

Editing

Ctrl + Space	Basic code completion (the name of any class, method or variable)
Ctrl + Shift + Space	Smart code completion (filters the list of methods and variables by expected type)
Ctrl + Shift + Enter	Complete statement
Ctrl + P	Parameter info (within method call arguments)
Ctrl + G	Quick documentation lookup
Shift + F1	External Doc
Ctrl + hover	Brief info
Ctrl + F1	Show descriptions of error or warning at caret
Alt + Insert	Generate code... (Getters, Setters, Constructors, hashCode/equal/s toString)
Ctrl + O	Override methods
Ctrl + I	Implement methods
Ctrl + Alt + T	Surround with... (if, else, try, catch, for, synchronized, etc.)
Ctrl + /	Comment/uncomment with line comment
Ctrl + Shift + /	Comment/uncomment with block comment
Ctrl + W	Select successively increasing code blocks
Ctrl + Shift + W	Decrease current selection to previous state
Alt + Q	Context info
Alt + Enter	Show intention actions and quick-fixes
Ctrl + Alt + L	Reformat code
Ctrl + Alt + O	Optimize imports
Ctrl + Alt + I	Auto-indent line(s)
Tab / Shift + Tab	Indent/unindent selected lines
Ctrl + X	Cut current line or selected block to clipboard
Ctrl + C	Copy current line or selected block to clipboard
Ctrl + V	Paste from clipboard
Ctrl + Shift + V	Paste from recent buffers...
Ctrl + D	Duplicate current line or selected block
Ctrl + Y	Delete line at caret
Ctrl + Shift + J	Smart line join
Ctrl + Enter	Smart line split
Shift + Enter	Start new line
Ctrl + Shift + U	Toggle case for word at caret or selected block
Ctrl + Shift + ⌘/⌥	Select till code block end/start
Ctrl + Delete/Backspace	Delete to word end/start
Ctrl + NumPad +/-	Expand/collapse code block
Ctrl + Shift + NumPad+	Expand all
Ctrl + Shift + NumPad-	Collapse all
Ctrl + F4	Close active editor tab

Usage Search

Alt + F7/Ctrl + F7	Find usages/Find usages in file
Ctrl + Shift + F7	Highlight usages in file
Ctrl + Alt + F7	Show usages

Navigation

Ctrl + N	Go to class
Ctrl + Shift + N	Go to file
Ctrl + Alt + Shift + N	Go to symbol
Alt + Right/Left	Go to next / previous editor tab
F12	Go back to previous tool window
Esc	Go to editor (from tool window)
Shift + Esc	Hide active or last active window
Ctrl + Shift + F4	Close active run / messages / find / ... tab
Ctrl + G	Go to line
Ctrl + E	Recent files popup
Ctrl + Alt + Left/Right	Navigate back / forward
Ctrl + Shift + Backspace	Navigate to last edit location
Alt + F1	Select current file or symbol in any view
Ctrl + B, Ctrl + Click	Go to declaration
Ctrl + Alt + B	Go to implementation(s)
Ctrl + Shift + I	Open quick definition lookup
Ctrl + Shift + B	Go to type declaration
Ctrl + U	Go to super-method / super-class
Alt + Up/Down	Go to previous / next method
Ctrl + ⌘/⌥	Move to code block end/start
Ctrl + F12	File structure popup
Ctrl + H	Type hierarchy
Ctrl + Shift + H	Method hierarchy
Ctrl + Alt + H	Call hierarchy
F2 / Shift + F2	Next/previous hover
F4 / Ctrl + Enter	Edit source / View
Alt + Home	Show navigation
F11	Toggle fullscreen
Ctrl + F11	Toggle fullscreen
Ctrl + #[0-9]	Go to tab
Shift + F11	Go to tab

Search, Replace

Double Shift	Search everywhere
Ctrl + F	Find
F3 / Shift + F3	Find next / Find previous
Ctrl + R	Replace
Ctrl + Shift + F	Find in path
Ctrl + Shift + R	Replace in path

Live Templates

Ctrl + Alt + J	Surround with Live Template
Ctrl + J	Insert Live Template
<i>iter</i>	Iteration according to Java 5/6/7/8
<i>inst</i>	Check object type with instanceof
<i>itc</i>	Iterate elements of java.util.Collection
<i>itr</i>	Iterate elements of java.util.Iterator
<i>itl</i>	Iterate elements of java.util.List
<i>psf</i>	public static final
<i>thr</i>	throw new

Refactoring

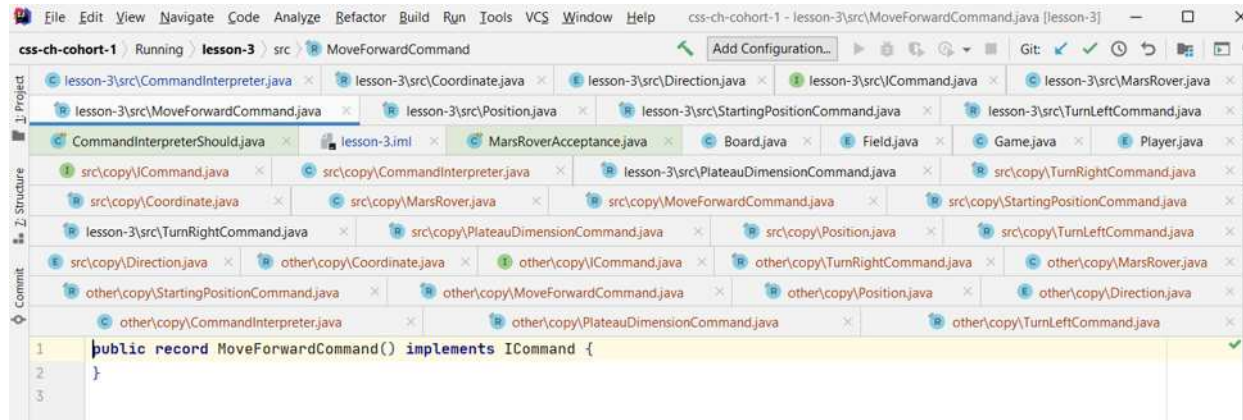
F5	Copy
F6	Move
Alt + Delete	Safe Delete
Shift + F6	Rename
Ctrl + F6	Change Signature
Ctrl + Alt + N	Inline
Ctrl + Alt + M	Extract Method
Ctrl + Alt + V	Extract Variable
Ctrl + Alt + F	Extract Constant
Ctrl + Alt + P	Extract Parameter

Debugging

F7	Run
F8	Step Over
F9	Step Into
F10	Step Out
F11	Toggle Breakpoint
Ctrl + F11	Toggle Breakpoint
Shift + F11	Toggle Breakpoint

Keep the Editor clean (1/2)

- For me tabs are like trees, at some point you can't see the forest for the trees...



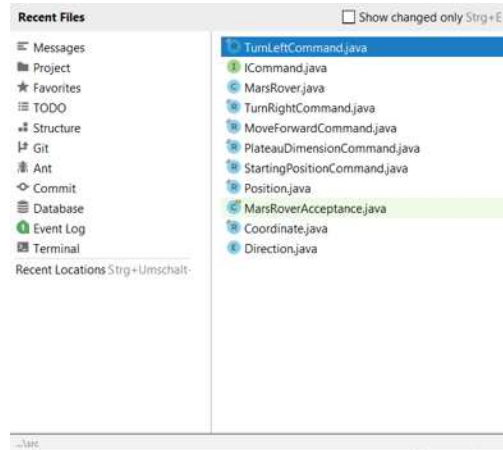
- What File do i currently work on?
- What was the file i worked on before?

Keep the Editor clean (2/2)

- Do we really need the tabs?



Use Recent Files instead of tabs **Ctrl + E**



Use Plugins (1/2)

Example:  *Rainbow Brackets*

Before:

```
private static Stream<Arguments> valueProvider() {  
    return Stream.of(  
        arguments("""  
                5 5  
                1 2 N  
                """,  
        List.of(new PlateauDimensionCommand(new Coordinate(x: 5, y: 5)),  
                new StartingPositionCommand(new Position(new Coordinate(x: 1, y: 2), Direction.NORTH))),  
        arguments("""
```

After:

```
private static Stream<Arguments> valueProvider() {  
    return Stream.of(  
        arguments("""  
                5 5  
                1 2 N  
                """,  
        List.of(new PlateauDimensionCommand(new Coordinate(x: 5, y: 5)),  
                new StartingPositionCommand(new Position(new Coordinate(x: 1, y: 2), Direction.NORTH))),  
        arguments("""
```



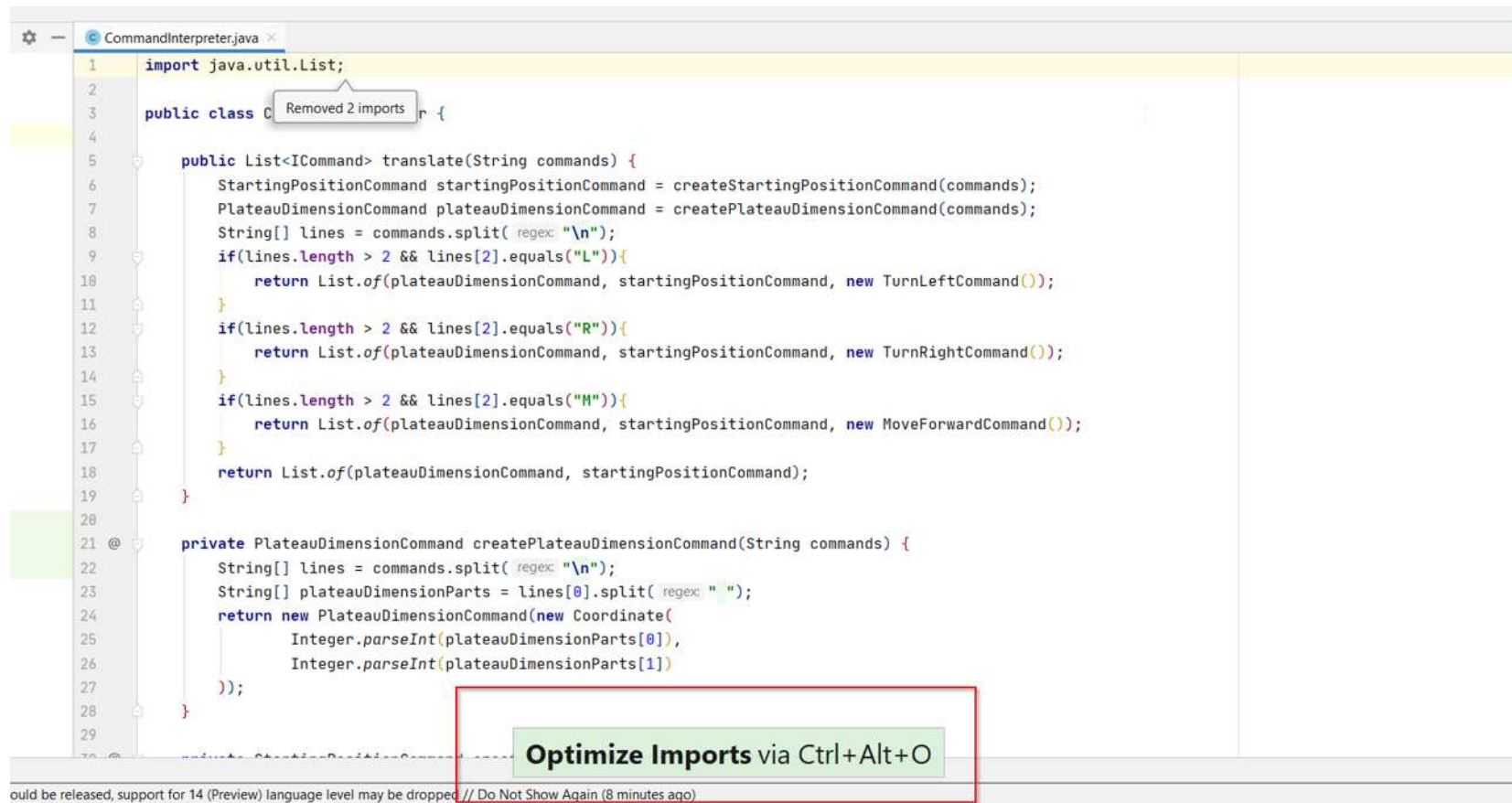
Ctrl + Shift + M

Move caret to matching brace



Use Plugins (2/2)

Example: *Presentation Assistant*



The screenshot shows a code editor window titled "CommandInterpreter.java" with the following Java code:

```
1 import java.util.List;
2
3 public class CommandInterpreter {
4
5     public List<ICommand> translate(String commands) {
6         StartingPositionCommand startingPositionCommand = createStartingPositionCommand(commands);
7         PlateauDimensionCommand plateauDimensionCommand = createPlateauDimensionCommand(commands);
8         String[] lines = commands.split( regex: "\n");
9         if(lines.length > 2 && lines[2].equals("L")){
10             return List.of(plateauDimensionCommand, startingPositionCommand, new TurnLeftCommand());
11         }
12         if(lines.length > 2 && lines[2].equals("R")){
13             return List.of(plateauDimensionCommand, startingPositionCommand, new TurnRightCommand());
14         }
15         if(lines.length > 2 && lines[2].equals("H")){
16             return List.of(plateauDimensionCommand, startingPositionCommand, new MoveForwardCommand());
17         }
18         return List.of(plateauDimensionCommand, startingPositionCommand);
19     }
20
21     private PlateauDimensionCommand createPlateauDimensionCommand(String commands) {
22         String[] lines = commands.split( regex: "\n");
23         String[] plateauDimensionParts = lines[0].split( regex: " ");
24         return new PlateauDimensionCommand(new Coordinate(
25             Integer.parseInt(plateauDimensionParts[0]),
26             Integer.parseInt(plateauDimensionParts[1])
27         ));
28     }
29
30     private StartingPositionCommand createStartingPositionCommand(String commands) {
31         return new StartingPositionCommand(new Coordinate(
32             Integer.parseInt(plateauDimensionParts[0]),
33             Integer.parseInt(plateauDimensionParts[1])
34         ));
35     }
36 }
```

A tooltip with the text "Optimize Imports via Ctrl+Alt+O" is visible over the code. A red box highlights this tooltip. At the bottom of the editor, a status bar message reads: "could be released, support for 14 (Preview) language level may be dropped // Do Not Show Again (8 minutes ago)".

Use multiple carets

- Clone caret: Press **Ctrl** twice, and then without releasing it, press up or down arrow keys.

```
7 }
8     return List.of(plateauDimensionCommand, startingPositionCommand);
9 }
10
11 @
12 private PlateauDimensionCommand createPlateauDimensionCommand(String command)
13 {
14     String[] lines = commands.split( regex: "\n");
15     String[] plateauDimensionParts = lines[0].split( regex: " ");
16     return new PlateauDimensionCommand(new Coordinate(
17         Integer.parseInt(plateauDimensionParts[0]),
18         Integer.parseInt(plateauDimensionParts[1])
19     ));
20 }
```

Clone Caret Below

- Add/Remove caret at mouse position: **Shift + Alt + MouseClick**

```
String[] lines = commands.split( regex: "\n");
if(lines.length > 2 && lines[2].equals("L")){
    return List.of(plateauDimensionCommand, startingPositionCommand);
}
if(lines.length > 2 && lines[2].equals("R")){
    return List.of(plateauDimensionCommand, startingPositionCommand);
}
if(lines.length > 2 && lines[2].equals("M")){
    return List.of(plateauDimensionCommand, startingPositionCommand);
}
```

- Select all occurrences: **Ctrl + Shift + Alt + J**

```
private PlateauDimensionCommand createPlateauDimensionCommand(String commands) {
    String[] lines = commands.split( regex: "\n");
    String[] plateauDimensionParts = lines[0].split( regex: " ");
    return new PlateauDimensionCommand(new Coordinate(
        Integer.parseInt(plateauDimensionParts[0]),
        Integer.parseInt(plateauDimensionParts[1])
    ));
}

private StartingPositionCommand createStartingPositionCommand(String commands) {
    String[] lines = commands.split( regex: "\n");
    String[] startingPositionLineParts = lines[1].split( regex: " ");
    return new StartingPositionCommand(
        Integer.parseInt(startingPositionLineParts[0]),
        Integer.parseInt(startingPositionLineParts[1])
    );
}
```

Select All Occurrences via Ctrl+Alt+Shift+J

Use build in features

Suppose you want to extend "**^(.+)+@(.+)\$**". What are you doing?

- Just extend it, because it's trivial
- Try and error?
- **Google**
- ...
- Let IntelliJ help you:

The image contains four screenshots illustrating the process of extending a regex in IntelliJ IDEA:

- Top-left:** A code editor snippet showing a Java class with a regex: `String regex = "^(.+)+@(.+)$";`. A context menu is open over the regex, with the option "Inject language or reference" highlighted.
- Top-right:** A search results window showing a list of languages. "RegExp (Regular Expression)" is selected and highlighted in blue.
- Bottom-left:** The same code editor snippet, but the context menu now shows "Check RegExp" as the selected option.
- Bottom-right:** A "RegExp" tester window. The "RegExp:" field contains `^(.+)+@(.+)$` and the "Sample:" field contains `p.miller@gmx.com`. The "Matches!" checkbox is checked, and the result shows a match.

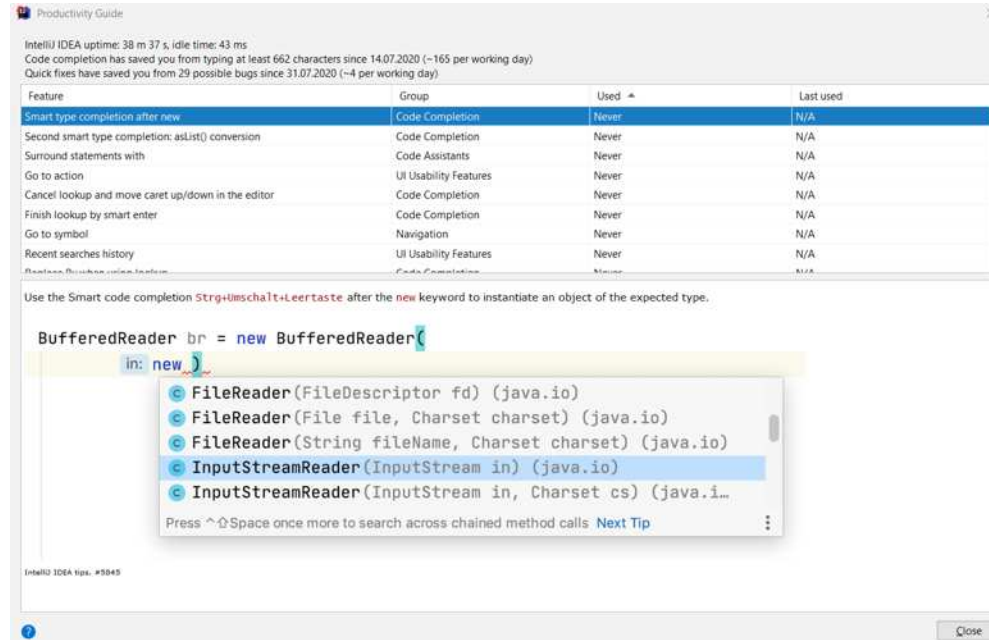


IntelliJ knows a lot of languages...



More

Productivity Guide



Complete Keymap:

<https://github.com/JetBrains/intellij-community/tree/master/platform/platform-resources/src/keymaps>

