a1a # tooling

IDE Basics: Getting Started with IntelliJ IDEA

TCSS 305 Programming Practicum

This guide introduces Integrated Development Environments (IDEs) and helps you get comfortable with IntelliJ IDEA, the tool you'll use throughout this course. You'll learn why IDEs matter for Java development and discover essential features that will make you more productive.

What is an IDE?

An **Integrated Development Environment** (IDE) is a software application that provides comprehensive tools for software development in one unified interface. The key word is *integrated*—instead of juggling separate programs for editing, compiling, running, and debugging code, an IDE brings everything together.

IDE vs Text Editor:

Feature	Text Editor	IDE
Code editing	Yes	Yes
Syntax highlighting	Often	Yes
Compiler integration	No	Yes
Debugger	No	Yes
Code completion	Limited	Advanced
Refactoring tools	No	Yes
Project management	No	Yes

A text editor like Notepad or VS Code (in basic mode) is fine for simple scripts. But for Java development—where you're managing multiple files, packages, imports, and complex builds—an IDE saves enormous amounts of time and catches errors before you even run your code.

> **Note**
>
> Think of the difference like cooking. A text editor is like having a knife and cutting board. An IDE is like having a fully equipped kitchen with appliances, measuring tools, and recipe organization all in one place.

Why IntelliJ IDEA Ultimate?

IntelliJ IDEA is the industry-standard IDE for Java development. It's created by JetBrains, a company that specializes in developer tools.

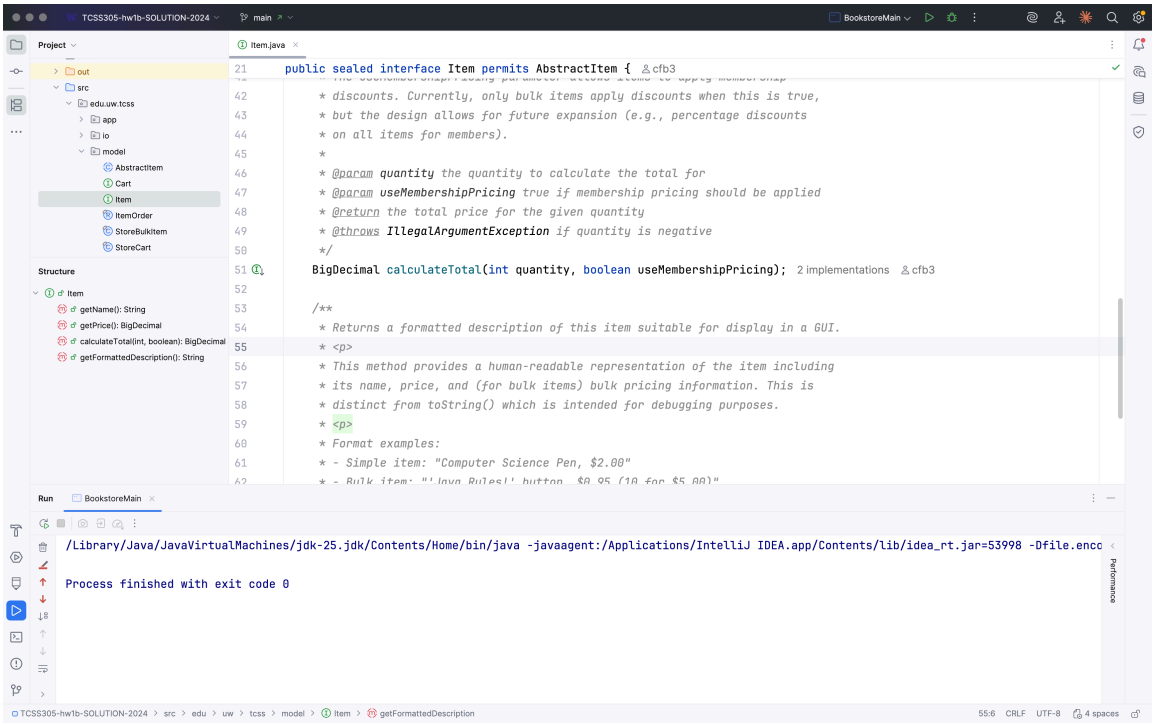
Why we use it in TCSS 305:

- **Industry standard** - Used by professional Java developers at companies like Google, Netflix, and Twitter
- **Free for students** - JetBrains provides free Ultimate licenses through their education program
- **Intelligent code assistance** - Goes far beyond basic autocomplete with deep understanding of your code
- **Built-in tools** - Version control (Git), testing, debugging, and code analysis all integrated
- **Plugin ecosystem** - Checkstyle-IDEA and other tools we use are available as plugins

> **Tip**
>
> If you haven't already, apply for your free student license at JetBrains Education. Use your UW email address for verification.

The IntelliJ Interface

When you first open IntelliJ with a project, you'll see several distinct areas. Here's how they're organized:



Key Areas

1. Project Panel (Left Side)

The Project panel shows your file tree—all the folders and files in your project. You'll see: - `src/` - Your Java source code - `test/` - Your unit tests - `files/` - Resource files (like inventory data) - Configuration files (`.iml` , `pom.xml` , etc.)

Double-click any file to open it in the editor.

2. Editor Area (Center)

This is where you write code. When you have multiple files open, they appear as tabs across the top. The editor provides: - Syntax highlighting (colors for keywords, strings, comments) - Real-time error detection (red underlines) - Code completion suggestions - Line numbers in the gutter (left margin)

3. Tool Windows (Bottom and Sides)

Tool windows provide additional functionality: - **Run** - Console output when you run your program - **Debug** - Debugging controls and variable inspection - **Terminal** - Command-line access - **Git** - Version control operations - **Checkstyle** - Linter results

Tool windows can be shown/hidden using **View > Tool Windows**.

> **About Linting**
>
> The Checkstyle tool window shows linting results—code style and quality issues that the compiler won't catch. See the Linters and Code Quality guide to learn what linters are and why they matter.

4. Run/Debug Toolbar (Top Right)

The green play button runs your program. The bug icon starts the debugger. The dropdown lets you select which configuration to run (useful when you have multiple main classes or test suites).

Essential Features

These are the features you'll use daily. Learn these shortcuts—they'll dramatically speed up your workflow.

Code Completion

Start typing and IntelliJ suggests completions based on context.

Action	Windows/Linux	macOS
Basic completion	`Ctrl+Space`	`Cmd+Space`
Smart completion	`Ctrl+Shift+Space`	`Cmd+Shift+Space`

Smart completion filters suggestions to only show what makes sense in the current context (e.g., only methods that return the expected type).

```
// Type "myPrice.set" and press Ctrl+Space
// IntelliJ suggests: setScale(), methods from BigDecimal
```

Quick Fixes (The Lightbulb)

When IntelliJ detects an issue, a lightbulb icon appears in the gutter. This is your gateway to automatic fixes.

Action	Windows/Linux	macOS
Show quick fixes	`Alt+Enter`	`Option+Enter`

Common quick fixes: - Add missing import statements - Create a method that doesn't exist yet - Implement interface methods - Surround with try/catch - Rename to match conventions

> **Tip**
>
> When you see a red squiggly underline, position your cursor there and press `Alt+Enter` (or `Option+Enter` on Mac). IntelliJ usually knows exactly how to fix the problem.

Navigation

Jump around your codebase quickly without scrolling through files.

Action	Windows/Linux	macOS
Go to definition	`Ctrl+Click` or `Ctrl+B`	`Cmd+Click` or `Cmd+B`
Find usages	`Alt+F7`	`Option+F7`
Go to file	`Ctrl+Shift+N`	`Cmd+Shift+O`
Go to class	`Ctrl+N`	`Cmd+O`
Recent files	`Ctrl+E`	`Cmd+E`

Go to definition is incredibly useful— `Ctrl+Click` on any method, class, or variable to jump directly to where it's defined, even if it's in a library.

Refactoring

Safely rename or restructure code. IntelliJ updates all references automatically.

Action	Windows/Linux	macOS
Rename	`Shift+F6`	`Shift+F6`

Action	Windows/Linux	macOS
Extract method	`Ctrl+Alt+M`	`Cmd+Option+M`
Extract variable	`Ctrl+Alt+V`	`Cmd+Option+V`

Rename is particularly valuable—if you rename a method, IntelliJ finds and updates every place it's called. No more find-and-replace mistakes.

Live Error Highlighting

IntelliJ compiles your code in the background as you type. Errors appear as red underlines immediately—you don't have to wait until you try to build.

- **Red underline** = compilation error (code won't compile)
- **Yellow underline** = warning (code will compile but may have issues)
- **Gray text** = unused code

The right-side stripe shows a summary of all issues in the file. Red marks are errors; yellow marks are warnings.

Running Your Code

Run Configurations

A **run configuration** tells IntelliJ how to run your code—which class to start, what arguments to pass, etc.

To create a run configuration: 1. Right-click on a class with a `main` method 2. Select **Run 'ClassName.main()'**

IntelliJ automatically creates a configuration for you. This configuration appears in the dropdown next to the Run button.

The Green Play Button

Once you have a configuration: - Click the **green play button** to run - Or press `Shift+F10` (Windows/Linux) or `Ctrl+R` (macOS)

Reading Console Output

When your program runs, output appears in the **Run** tool window at the bottom. This shows: - Anything your program prints (via Logger or System.out) - Error messages if something goes wrong - The exit code when the program finishes

> **Warning**
>
> In TCSS 305, avoid using `System.out.println()` in production code. Use the provided Logger instead. See the assignment instructions for details.

Quick Debugging

When your code doesn't behave as expected, the debugger helps you understand what's happening step by step.

Setting Breakpoints

Click in the **gutter** (the margin left of line numbers) to set a **breakpoint**—a red dot appears. When you run in debug mode, execution pauses at this line.

Debug Mode vs Run Mode

Action	Windows/Linux	macOS
Run (normal)	`Shift+F10`	`Ctrl+R`
Debug	`Shift+F9`	`Ctrl+D`

Or click the **bug icon** instead of the play button.

Stepping Through Code

Once paused at a breakpoint:

Action	Windows/Linux	macOS	Description
Step Over	`F8`	`F8`	Execute current line, move to next
Step Into	`F7`	`F7`	Enter the method being called

Action	Windows/Linux	macOS	Description
Step Out	`Shift+F8`	`Shift+F8`	Finish current method, return to caller
Resume	`F9`	`Cmd+Option+R`	Continue until next breakpoint

Variables Panel

While debugging, the **Variables** panel shows the current values of all variables in scope. This is invaluable for understanding why your code is behaving unexpectedly.

You can also hover over any variable in the editor to see its current value.

> **Tip**
>
> Start simple: set one breakpoint, run in debug mode, and practice stepping through a few lines. Once you're comfortable with Step Over and the Variables panel, you'll have a powerful tool for understanding your code.

AI Code Completion

IntelliJ includes AI-powered code suggestions that can complete entire lines or suggest code blocks.

Learning Considerations

> **Caution**
>
> While AI assistance can be helpful, over-relying on it can hinder your learning. As a student, you need to build foundational skills by writing code yourself and understanding why it works.

Recommendations for learning: - Focus on understanding the code you write, not just accepting suggestions - If AI suggests something, make sure you can explain what it does - Consider disabling full-line AI completions while learning fundamentals - Use AI for boilerplate code (getters, equals methods) once you understand the patterns

Accessing AI Settings

To configure AI features: 1. Go to **File > Settings** (Windows/Linux) or **IntelliJ IDEA > Preferences** (macOS) 2. Navigate to **Editor > General > Inline Completion** 3. Adjust settings as desired

You can also access AI settings through **Tools > AI Assistant** if available in your version.

Common Gotchas for New Users

"Cannot resolve symbol"

Problem: IntelliJ shows red text and says it can't find a class, method, or variable.

Common causes and fixes:

1. **Missing import** - Press `Alt+Enter` and select "Import class"
2. **File not in src/ folder** - Java files must be in the correct source directory
3. **Wrong package declaration** - The package in your code must match the folder structure
4. **Project not loaded correctly** - Try **File > Invalidate Caches / Restart**

Wrong JDK Selected

Problem: Your project uses Java features that IntelliJ doesn't recognize, or you get unexpected compilation errors.

Fix: Verify the JDK version: 1. Go to **File > Project Structure** (or `Ctrl+Alt+Shift+S` / `Cmd+;`) 2. Under **Project**, check the **SDK** dropdown 3. Ensure it shows JDK 25 (or the version required for TCSS 305) 4. Under **Modules**, verify each module uses the same SDK

Code Changes Don't Take Effect

Problem: You edited code but running the program shows old behavior.

Fixes: 1. **Save files** - `Ctrl+S` / `Cmd+S` (though IntelliJ auto-saves) 2. **Rebuild** - **Build > Rebuild Project** 3. **Check run configuration** - Make sure you're running the right class

IntelliJ Feels Slow

Problem: The IDE is laggy or unresponsive.

Fixes: 1. **Close unused projects** - Each open project uses memory 2. **Increase memory** - **Help > Change Memory Settings** 3. **Disable unused plugins** - **Settings > Plugins** 4. **Invalidate**

caches - **File > Invalidate Caches / Restart**

Summary

IntelliJ IDEA is a powerful tool that will become second nature with practice. Start with these essentials:

What to Learn	Why It Matters
Code completion (`Ctrl+Space`)	Write code faster with fewer typos
Quick fixes (`Alt+Enter`)	Let IntelliJ fix common problems automatically
Go to definition (`Ctrl+Click`)	Navigate large codebases efficiently
Rename refactoring (`Shift+F6`)	Safely rename without missing references
Basic debugging (`F8` to step)	Understand what your code actually does

Don't try to learn everything at once. Master these basics, then gradually explore more features as you need them.

Further Reading

External Resources

- IntelliJ IDEA Documentation - Official getting started guide
- IntelliJ IDEA Keyboard Shortcuts - Complete shortcut reference
- JetBrains Academy - Free interactive courses including IDE skills
- IntelliJ IDEA Tips & Tricks - Official YouTube playlist

References

Primary Texts:

- Horstmann, C. S. (2022). *Core Java, Volume I: Fundamentals* (12th ed.). Oracle Press. Chapter 2: The Java Programming Environment.
- Reges, S., & Stepp, M. (2020). *Building Java Programs* (5th ed.). Pearson. Appendix A: Working with an IDE.

Tooling Documentation:

- JetBrains IntelliJ IDEA Documentation — Official IDE reference
- IntelliJ IDEA Getting Started — Beginner tutorials
- IntelliJ IDEA Keyboard Shortcuts Reference — Complete shortcut list
- IntelliJ IDEA Debugging — Debugger documentation

Language Documentation:

- Oracle JDK 25 Documentation — Official Java SE reference

This guide is part of TCSS 305 Programming Practicum, School of Engineering and Technology, University of Washington Tacoma.