JAVAWORLD a IDE

Compare Eclipse, NetBeans, and IntelliJ IDEA for features, usability, and project size and type

By Martin Heller
Contributing Editor, JavaWorld
DEC 18, 2018 9:56 AM PST

Updated: December 2018.

Every Java developer needs a programming editor or IDE that can assist with the grungier parts of writing Java and using class libraries and frameworks. Deciding which editor or IDE will best suit you depends on several things, including the nature of the projects under development, your role in the organization, the process used by the development team, and your level and skills as a programmer. Additional considerations are whether the team has standardized on tools, and your personal preferences.

The three IDEs most often chosen for server-side Java development are IntelliJ IDEA, Eclipse, and NetBeans. These aren't the only choices, however, and this review will include some lightweight IDEs as well.

For this roundup, I did fresh installations of IntelliJ IDEA Ultimate 2018.3, Eclipse IDE 2018-09 for Java EE Developers, and Apache NetBeans (incubating) IDE 9 on a Mac. I also checked out several open source Java projects so that I could test all of the IDEs on the same projects.

About this update

This IDE review was first published in September 2016, and has been updated in December 2018. In those intervening years the Java language, APIs, JVM ecosystem, and some frameworks have evolved significantly. Java EE 8 introduced or updated many Java technology specifications, including JSON-B (JavaScript Object Notation Binding), Java EE Security, Servlet 4.0, and JSF (JavaServer Faces) 2.3 for building server-side user interfaces. Java EE 8 was also the final Java enterprise release from Oracle: The Eclipse Foundation has taken charge of stewarding the technology, which it has rebranded to Jakarta EE. Meanwhile, JUnit has advanced to version 5, breaking integrations; IDEA and Eclipse both have native support for JUnit 5, but as of this writing NetBeans does not.

All of these changes should be part of your evaluation of an IDE, whether for general use or for a particular project.

NetBeans 10 adds support for JUnit 5 and JDK 11

Released in January 2019, NetBeans 10 adds support for JDK 11 and JUnit 5.

Basics: What you need from a Java IDE

At minimum, you would hope that your IDE supports Java 8 and/or 11 (the LTS versions), Scala, Groovy, Kotlin, and any other JVM languages you regularly use. You'd also want it to support the major application servers and the most popular web frameworks, including Spring MVC, JSF, Struts, GWT, Play, Grails, and Vaadin. Your IDE should be compatible with whatever build and version control systems your development team uses; examples include Apache Ant with Ivy, Maven, and Gradle, along with Git, SVN, CVS, Mercurial, and Bazaar. For extra credit, your IDE should be able to handle the client and database layers of your stack, supporting embedded JavaScript, TypeScript, HTML, SQL, JavaServer Pages, Hibernate, and the Java Persistence API.

Finally, you would hope that your Java IDE lets you edit, build, debug, and test your systems with ease and grace. Ideally, you'd not only have intelligent code completion, but refactoring and code metrics. If you're in a shop that does test-driven development, you want support for your testing frameworks and stubbing. If your group uses a ticket system and CI/CD, it's best if your IDE can connect to them. If you need to deploy to and debug on containers and clouds, your IDE should help you do so.

With that foundation in mind, let us consider the contenders.

IntelliJ IDEA

IntelliJ IDEA, the premier Java IDE in terms of both features and price, comes in two editions: the free Community edition, and the paid Ultimate edition, which has additional features.

The Community edition is intended for JVM and Android development. It supports Java, Kotlin, Groovy, and Scala; Android; Maven, Gradle, and SBT; and Git, SVN, Mercurial, CVS, and TFS.

The Ultimate edition, intended for web and enterprise development, supports Perforce in addition to the other version control systems; supports JavaScript and TypeScript; supports Java EE, Spring, GWT, Vaadin, Play, Grails, and other frameworks; and includes database tools and SQL support.

The idea is that the commercial (Ultimate) edition will earn its place on a professional's desktop, justifying a paid subscription through increased programmer productivity. If you are earning $50K-$100K per year as a Java developer, it doesn't take much of a productivity boost to give you a quick ROI on a $500/year business IDEA subscription. The price goes down in subsequent years for businesses, is much lower for startups and individuals, and is free for students, teachers, "Java champions," and open source developers.

IntelliJ touts IDEA for deep insight into your code, developer ergonomics, built-in developer tools, and a polyglot programming experience. Let's drill down and see what these features mean, and how they can help you.

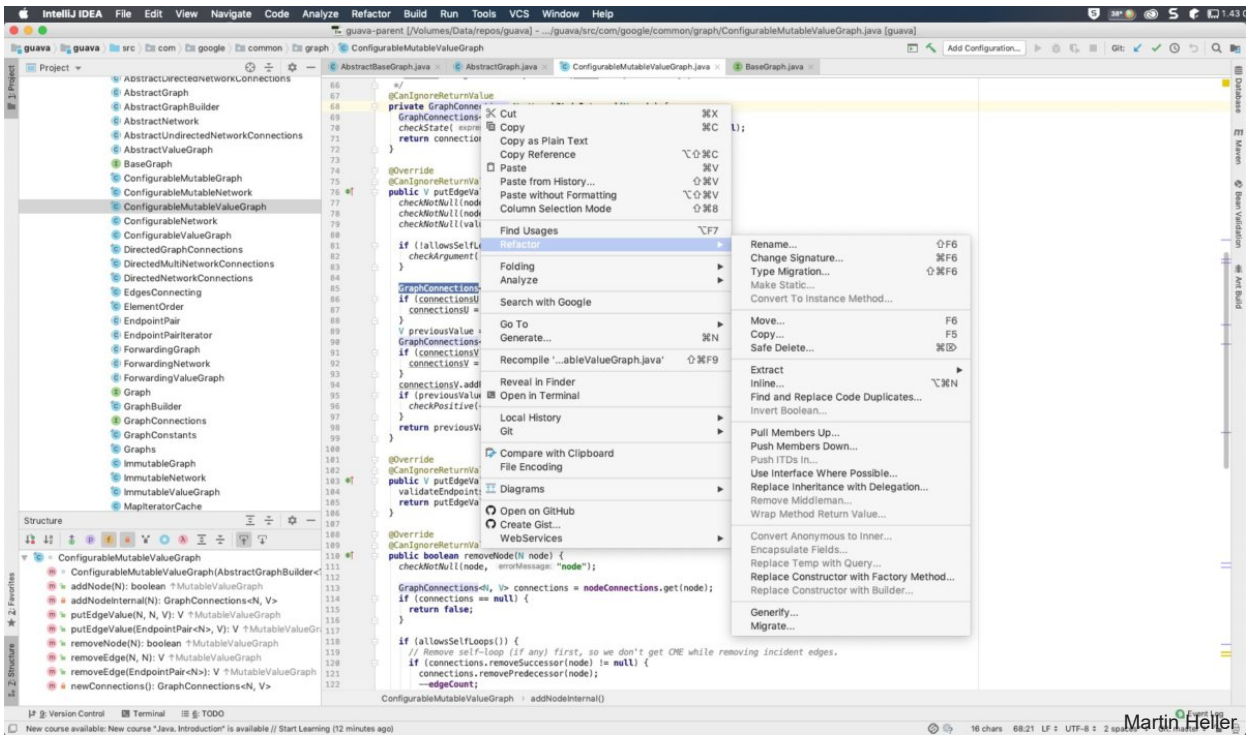

Figure 1. IntelliJ IDEA offers a wide variety of context-sensitive actions within the editor, including an extensive selection of refactoring options and integration with GitHub.

Deep insight into your code

Syntax coloring and simple code completion are a given for Java editors. IDEA goes beyond that to provide "smart completion," meaning that it can pop up a list of the most relevant symbols applicable in the current context. These are ranked by your personal frequency of use. "Chain completion" goes deeper and displays a list of applicable symbols *accessible via methods or getters* in the current context. IDEA also completes static members or constants, automatically adding any needed import statements. In all code completions, IDEA tries to guess the runtime symbol type, refine its choices from that, and add class casts as needed.

Java code often contains other languages as strings. IDEA can inject fragments of SQL, XPath, HTML, CSS, and/or JavaScript code into Java String literals. For that matter, it can refactor code across multiple languages; for example, if you rename a class in a JPA statement, IDEA will update the corresponding entity class and JPA expressions.

When you're refactoring a piece of code, one of the things you typically want to do is also refactor all the duplicates of that code. IDEA Ultimate can detect duplicates and similar fragments and apply the refactoring to them as well.

IntelliJ IDEA analyzes your code when it loads, and when you type. It offers inspections to point out possible problems and, if you wish, a list of quick fixes to the detected problem.

Developer ergonomics

IntelliJ designed IDEA with the developer's creative flow--*aka* "being in the zone"--in mind. The Project tool window shown at the left in Figure 1 disappears from view with a simple mouse click, so that you can concentrate on the code editor. Everything you want to do while editing has a keyboard shortcut, including bringing up symbol definitions in a pop-up window. While learning the shortcuts does take time and practice, eventually they become second nature. Even without knowing the shortcuts, a developer can learn to use IDEA easily and quickly.

The design of the IDEA debugger is especially nice. Variable values show up right in the editor window, next to the corresponding source code. When the state of a variable changes, its highlight color changes as well.

Built-in developer tools

IntelliJ IDEA provides a unified interface for most major version control systems, including Git, SVN, Mercurial, CVS, Perforce, and TFS. You can do all your change management right in the IDE. As I tested IDEA, I wished that the *last change* in a source code block would show up in the editor window as an annotation (like it does in Visual Studio). As it turns out, there's a plugin for that.

IDEA also integrates build tools, test runners, and coverage tools, as well as a built-in terminal window. IntelliJ doesn't have its own profiler, but it supports several third-party profilers through plugins. These include YourKit, created by a former IntelliJ lead developer, and VisualVM, which is a repackaged version of the NetBeans profiler.

Debugging Java can be a pain when mysterious things happen in classes for which you have no source code. IDEA comes with a decompiler for those cases.

Java server programming often involves working with databases, so IDEA Ultimate includes SQL and NoSQL database tools. If you need more, a dedicated SQL IDE (DataGrip) is available as part of an all-products subscription that's only a little more expensive than an IDEA Ultimate subscription.

IntelliJ IDEA supports all the major JVM application servers, and can deploy to and debug in the servers, fixing a major pain point for Enterprise Java developers. IDEA also supports Docker through a plugin that adds a Docker tool window. (Speaking of plugins, IntelliJ has a lot of them.)

Polyglot programming

IDEA has extended coding assistance for Spring, Java EE, Grails, Play, Android, GWT, Vaadin, Thymeleaf, Android, React, AngularJS, and other frameworks. Not all of these are Java frameworks. In addition to Java, IDEA understands many other languages out of the box, including Groovy, Kotlin, Scala, JavaScript, TypeScript, and SQL. If you need more, there currently are hundreds of IntelliJ language plugins, including plugins for R, Elm, Go, Rust, and D.

Eclipse IDE

Eclipse, long the most popular Java IDE, is free and open source and is written mostly in Java, although its plugin architecture allows Eclipse to be extended in other languages. Eclipse originated in 2001 as an IBM project to replace the Smalltalk-based IBM Visual Age family of IDEs with a portable Java-based IDE. A goal of the project was to eclipse Microsoft Visual Studio, hence the name.

Java's portability helps Eclipse be cross-platform: Eclipse runs on Linux, Mac OS X, Solaris, and Windows. The Java Standard Widget Toolkit (SWT) is at least partially responsible for Eclipse's look and feel, for good or ill. Likewise, Eclipse owes its performance (or, some say, lack thereof) to the JVM. Eclipse has a reputation for running slowly, which harks back to older hardware and older JVMs. Even today it can feel slow, however, especially when it is updating itself in the background with many plugins installed.

Part of the overhead going on in Eclipse is its built-in incremental compiler, which runs whenever it loads a file and whenever you update your code. This is on balance a very good thing, and provides error indicators as you type.

Independent of the build system, an Eclipse Java project also maintains a model of its contents, which includes information about the type hierarchy, references, and declarations of Java elements. This is also on balance a good thing, and enables several editing and navigation assistants as well as the outline view.

The current version of Eclipse is 2018-09. I installed the Eclipse IDE for Java EE Developers, but there are many other installation packages, including the option to install the minimal Eclipse SDK and add plugins only as needed. The last option is not for the faint of heart, however: it's not hard to introduce conflicts between plugins that didn't actually *say* they were incompatible.

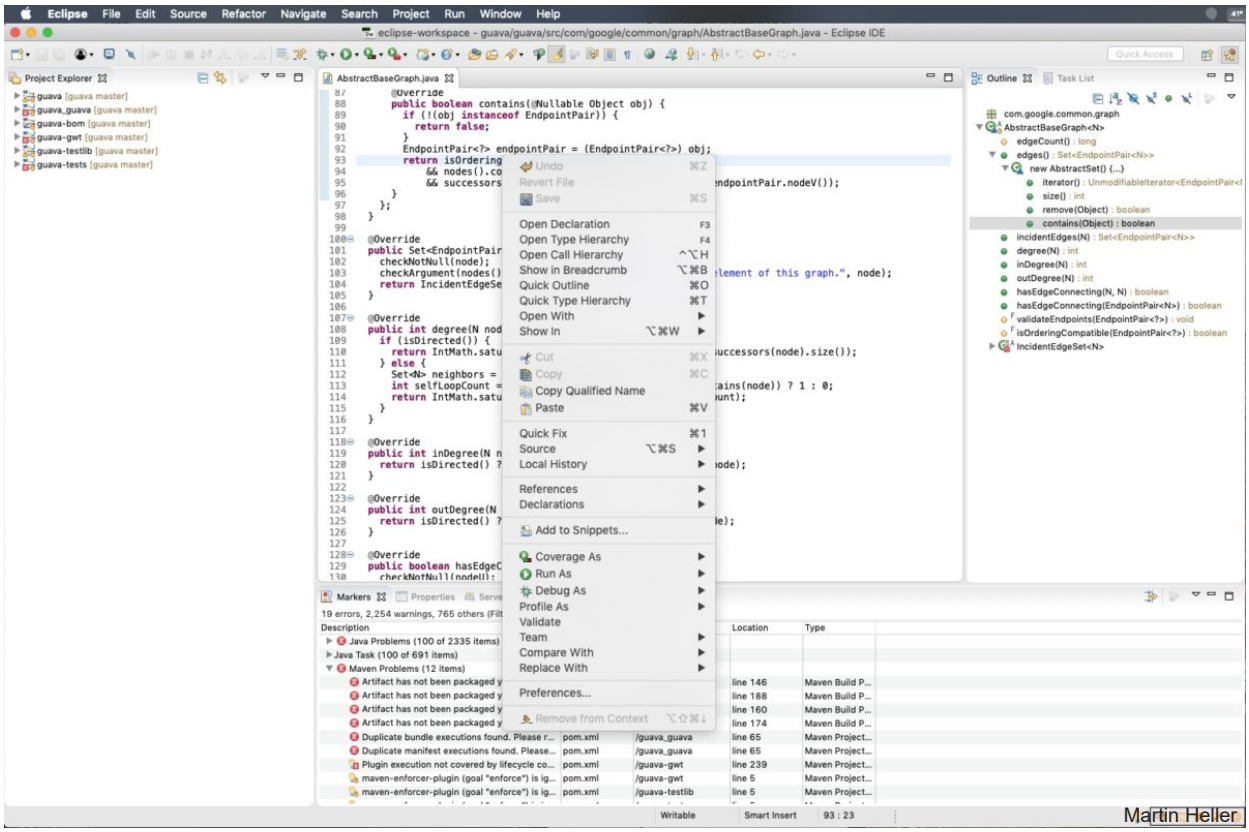

Figure 2. Clockwise from the top left, we're seeing four panes in the Eclipse workbench: the Project Explorer, the Java editor, the Java class outline, and the problems and tasks lists.

Extensible tools support

The plugin ecosystem is one of Eclipse's strengths, as well as being a source of occasional frustration. The Eclipse marketplace contains over 1,600 solutions currently, and community-contributed plugins may or may not work as advertised. Still, Eclipse plugins include support for over 100 programming languages and almost 200 application development frameworks.

Most Java servers are also supported: if you define a new server connection from Eclipse, you'll come to a list of vendor folders, underneath which you'll find about 30 application servers, including nine versions of Apache Tomcat. The commercial vendors tend to lump their offerings together: for example, there is only one item under Red Hat JBoss Middleware, which includes WildFly and EAP Server Tools, as well as JBoss AS.

Editing, browsing, refactoring, and debugging

A developer's first experience with Eclipse can be disconcerting, even confusing. This is because your first task is to adapt to Eclipse's conceptual architecture of workspaces, perspectives, and views, the functions of which are determined by what plugins you have

installed. For Java server development, for example, you are likely to use the Java, Java EE, and Java browsing perspectives; the package explorer view; the debugging perspective; a team synchronizing perspective; web tools; a database development perspective; and a database debugging perspective. In practice, all of those will start to make sense once you open the views you need.

There is often more than one way to do a given task in Eclipse. For example, you can browse code with the project explorer and/or the Java browsing perspective; which you choose is a matter of taste and experience.

Java searching support allows you to find declarations, references, and occurrences of Java packages, types, methods, and fields. You can also use Quick Access to search, and use quick views to pop up things like class outlines.