JAVAWORLD

FEATURE

Choosing your Java 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

Adding methods and generating classes are supported by error annotations and content assist. Common code patterns can be generated from code templates, and Eclipse can automatically generate and organize your import statements. Refactoring Java in Eclipse supports 23 operations, ranging from common renaming operations to more obscure transformations right out of Martin Fowler's book. Refactoring can be performed not only interactively, but also from refactoring scripts.

Eclipse supports debugging both locally and remotely, assuming that you are using a JVM that supports remote debugging. Debugging is fairly standard: you typically set breakpoints, and then view variables in a tab of the debugging perspective. You can of course step through your code and evaluate expressions.

Eclipse has extensive help and documentation, of varying age, currency, and utility. It's not unusual to discover that the documentation includes images that don't match the current version of the software, or that the keystrokes for your operating system are different from the ones called out in the help. I'm afraid it's one of the common problems with open source projects: the documentation can lag the software by months or even years. Because the ecosystem is so big, Eclipse has more than its share of documentation issues.

NetBeans

The NetBeans Java IDE started life as a university student project in Prague in 1996, became a commercial product in 1997, was bought by Sun in 1999, and was released to open source in 2000. It was recently donated to Apache, and has been accepted as an incubating project.

Oracle is turning over the code in chunks; the initial drop was only the Java SE portion of the code. You need to use the 8.2 libraries to get the other plugins you may be used to having. The turmoil in NetBeans land during the hand-off may explain why NetBeans 9 doesn't yet support JUnit 5: at this point that integration is only in the outline phase.

The current version, 9.0, lacks the system-specific installers we expected in previous versions. I downloaded the binaries and unpacked them. I quickly discovered the need to install `nb-java`, as this version doesn't run on Java 8. It does have good support for JDK 8, 9, and 10 code, however. Its editors, code analyzers, and converters can help you to upgrade your applications to use new Java language constructs, such as lambdas, functional operations, method references, and the `var` type.

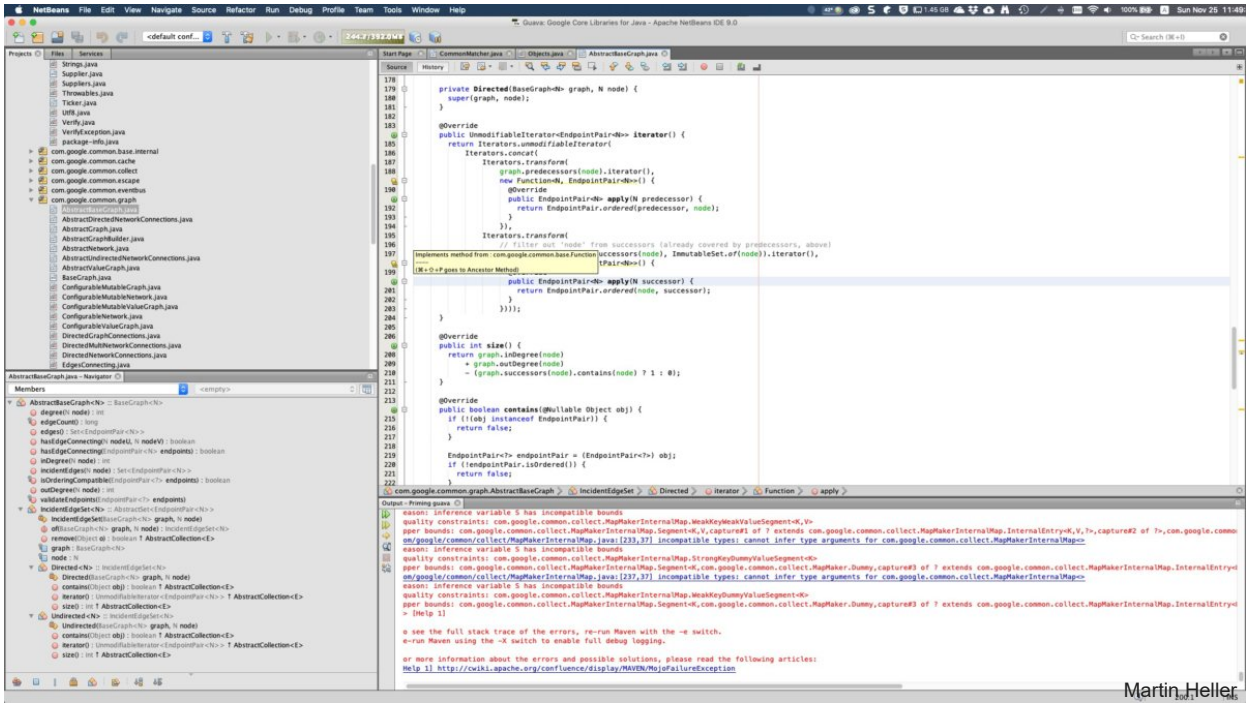

Figure 3. This is NetBeans 9 showing, clockwise from the upper left, the Projects, editor, output, and Navigator windows. The errors in the output were caused by NetBeans 9's native lack of Maven support, and were fixed when I added the Groovy kit from the NetBeans 8.2 repository.

Editing and refactoring

The language-aware NetBeans editor detects errors while you type and assists you with documentation popups and smart code completion. It seems to do so with fewer noticeable pauses than Eclipse does, although slightly more than IntelliJ IDEA. NetBeans also offers a full range of refactoring tools to allow you to restructure code without breaking it; performs source code analysis; and offers an extensive set of hints to quickly fix or enhance your code. NetBeans includes a design tool for Swing GUIs, previously known as "Project Matisse."

The Inspect & Transform tool enables you to run inspections across your codebase while automatically fixing your code. Personally, I always make sure I've checked in all my code and run all my unit tests successfully before running tools that can make sweeping changes; I've been burnt more than once by automatic "fixes" that cause regressions.

Building, debugging, and profiling

NetBeans has good built-in support for Maven and Ant, and a plugin for Gradle. I was pleased to discover that existing Maven projects are now treated as "native," meaning that you simply open them rather than importing them. NetBeans also includes a sexy (and useful) graph view for Maven dependencies.

The NetBeans Java debugger is good, albeit conventional. A separate visual debugger lets you take GUI snapshots and visually explore the GUI of JavaFX and Swing applications. The NetBeans profiler is very nice for understanding both CPU and memory use, and has good tools for finding memory leaks.

Comparing the big three Java IDEs

I personally have used Eclipse, NetBeans, and IntelliJ IDEA over the years, in that chronological order. After each switch, I felt that I had improved my productivity once I got used to the new IDE. Even once I thought I had firmly switched to IntelliJ, however, there were times I had to return to one of the other Java IDEs, for example during the period when Eclipse was the only IDE supported for Android development. (Android Studio, the current official Android IDE, is based on IntelliJ IDEA.)

I have full-time Java developer friends who use and swear by each of the big three IDEs. The IntelliJ IDEA users in particular are as loyal to their IDE as Visual Studio C++ and C# coders are to theirs, and claim that their productivity gains returned the cost of their annual subscriptions within a few weeks of use. NetBeans and Eclipse users are almost as loyal to their choices, and some wonder why people pay money for IntelliJ.

I tend to recommend that new Java coders *not* use Eclipse. Even though it's the most popular Java IDE, it has the steepest learning curve and the most potential for confusion, both in daily use and when maintaining the IDE. The many perspectives and views offer all sorts of functionality, but switching from one perspective to another can be jarring and disturb your flow.

Eclipse has the largest plugin ecosystem of any IDE, and also the greatest tendency to become unusable if you install an incompatible set of plugins. Sadly, I've had to delete my broken Eclipse installation and start over with an official distribution bundle at least half a dozen times over the years. At this point, I always start fresh when a new Eclipse "release train" comes out in June.

NetBeans is good enough for most people, and has a nice profiler. I use it in a pinch, but I still prefer IntelliJ IDEA Ultimate.

For new Java coders without a tools budget, the choice is between NetBeans and IntelliJ IDEA Community Edition. If you're coding Java servers with little or no money to spare, then NetBeans might be the better choice, unless you fall into a category that would entitle you to a free or discounted copy of IntelliJ IDEA Ultimate–such as being a student or working on an open source project.

Lightweight Java IDEs

While most people do best developing Java with IntelliJ, NetBeans, or Eclipse, there are still cases where you might want a lightweight IDE, or even a programming editor, such as Sublime Text, emacs, or vim, with a Java plugin.

If you’re looking for a lightweight IDE to try, consider these reasonable options:

- **DrJava** is a free lightweight development environment for writing Java programs. Designed for student use at Rice University, it currently has over three million downloads. DrJava is intended to foster test-driven software development. It includes an intelligent program editor, an interactions pane for evaluating program text, a source level debugger, and a unit-testing tool.
- **BlueJ** is a free Java development environment designed for beginners at Kings College London. It is supported by Oracle. BlueJ has a deliberately smaller and simpler interface than professional environments like NetBeans or Eclipse, and there is an introductory college textbook about learning OOP with BlueJ.
- **JCreator** is a lightweight Java IDE for Windows, written in C++ for performance reasons. The paid Pro version has a debugger, Ant support, and code wizards; the free LE version does not. JCreator does not seem to have been updated since 2015, however.
- **Eclipse Che** is a browser- and cloud-based IDE and developer workspace server. Che supports Java along with C++, JavaScript, Python, PHP, Ruby, and SQL.

Project-based tips for choosing a Java IDE

I've described the notable features of each of the top three Java IDEs and offered a glimpse at a handful of lightweight alternatives. You'll need to weigh this information against your personal development needs and resources to decide which IDE will suit you best. In addition to personal considerations, there are also project-based considerations. In many cases, it is easiest to use the same IDE as the rest of your development team, but that isn't completely necessary.

For example, if a team project is hosted on GitHub, then your life will be easier if your IDE supports GitHub. That isn't an absolute requirement, however: you can always use a GitHub client or git command-line and switch back and forth to your IDE. On the other hand, you really do want your IDE to support whatever build system has been adopted by the team. If it's Maven, for example, you don't want to have to reinvent the build system in Ant for your local testing. Fortunately, the big three Java IDEs all support Ant, Maven, and Gradle, either out of the box or with a plugin. That isn't necessarily the case with the lightweight IDEs.

You are going to want to your IDE to support the JRE version that is standard for the project; if there's a version mismatch, you *will* run into bugs that the rest of the team can't reproduce. That's not a situation you want to create. Fortunately, JRE mismatches are more often configuration errors than errors caused by lack of support in the IDE: the exceptional case happens briefly when an IDE hasn't yet released an update for a new Java version.

It helps a great deal if your IDE has strong support for the frameworks and technologies used in your project. You can get by without that, but if, for example, the IDE knows how JPA statements relate to entity classes and JPA expressions (as IntelliJ does), then you are likely to spend less time on your JPA-related code. And if the IDE supports the testing framework and test runner used by the project, you'll be able to test without leaving your workspace.

You'll also want your IDE to support deploying your Java apps to containers, specifically Docker containers with the option to work with Kubernetes orchestration. Direct support for the public clouds you use will make your life much simpler. And if your IDE can support JVM-based frameworks for machine learning and deep learning, such as Spark.ML and H2O.ai, that's a definite plus.

Finally, it helps if your IDE can connect with whatever ticketing and bug tracking system has been adopted for the project. Again, you can get by using a standalone or web client for, say, JIRA, but you'll be more productive if you can check out your tickets directly from your IDE.

Conclusion

I've made a strong case for IntelliJ IDEA Ultimate, which many would consider the Cadillac of modern Java IDEs. While it's not free like Eclipse or NetBeans, I believe the productivity gain is worth the annual subscription. For developers just starting out, or those preferring not to pay, I recommend NetBeans over Eclipse. Whereas Eclipse's plugin ecosystem once made it the top choice for developers, today it has become unwieldy and somewhat poorly maintained.

I also touched on lightweight alternatives, including two designed for student use. These are worth experimenting with, and could be your best option if you are just learning Java, if you find the full-featured IDEs overwhelming, or you just like a lighter weight development environment.

Martin Heller is a contributing editor and reviewer for InfoWorld. Formerly a web and Windows programming consultant, he developed databases, software, and websites from 1986 to 2010. More recently, he has served as VP of technology and education at Alpha Software and chairman and CEO at Tubifi.

Follow