PART 1

Creating and Managing Projects

CHAPTER 1

Integrated Development Environments

In the beginning, code was written using simple text-based tools like Notepad. For the purposes of this discussion, I'll define "beginning" as the early to mid-1990s, when Java first started to become popular. Using the combination of a text editor and command prompt, users could write and compile code.

It was quickly determined that this approach did not provide the most efficient development environment. For example, if you made a code syntax mistake in the text editor, there was no way to identify the problem until you saved and compiled the file. You would then review the compilation error, locate the offending line in the code, and attempt to determine the cause. Compilation errors are not always entirely helpful in diagnosing the problem with your code.

Many novice programmers start out using the Notepad and command-prompt environment. There is nothing inherently wrong with this approach, since some professionals still do the same thing. For an absolute beginner learning Java, using a plain text editor can sometimes be the easiest and fastest approach. However, text editors do not provide assistance with language syntax, compiler integration, intelligent refactoring support, or other code-writing capabilities.

One of the useful features most text editors possess is called Find and Replace. Using this simple capability, programmers could replace occurrences of a word or phrase with another. This worked for certain situations, but could cause problems. Suppose you created the following class:

```
public class SomeCode {

    public void myMethod1(String var) {

        String FirstName = var.toUpperCase();

        // do something with FirstName
    }
    public void myMethod2(String var) {

        String FirstName = var.toLowerCase();

        // do something else with FirstName
    }

}
```

The SomeCode class includes two methods: myMethod1 and myMethod2. If you later needed to rename the FirstName variable in myMethod1, you could manually edit each line of code to alter the name. Obviously, this is a simple example, but if myMethod1 happened to be a hundred lines long and FirstName appeared in many places, then manual editing of the code could take quite a long time. You could also use the text editor's Find and Replace functionality to quickly replace all occurrences of FirstName with the new variable name. However, the original change request specified only the FirstName variable in the myMethod1 method and *not* in the myMethod2 method. Using Find and Replace could incorrectly replace the wrong occurrences of FirstName in myMethod1 and myMethod2. Of course, it's possible to replace occurrences one by one, but that can take time and be prone to human error.

Some text editors offer more advanced support for programming languages. The popular Unix-based tool Emacs offers many interesting features. Emacs offers advanced text matching and replacement capabilities. Through plug-ins, it can also provide Java syntax highlighting, code indentation, basic debugging, and compilation support. These are great pieces of functionality, but still do not offer the most flexible and productive environment.

Why Use an IDE?

The first question anyone who uses Emacs or text editors might ask is, "Why use an IDE?" Some programmers tend to grow attached to a specific tool set or programming language and are resistant to change. An important quality in today's ever-changing world is the ability to adapt to new technology.

New tool sets can help professional programmers in many ways. A programmer's time should be spent writing code, rewriting code, and testing code. You shouldn't need to waste time trying to figure out how to rename methods across your code, generate project documentation, or correctly compile all the classes in a package. Once you have identified the action you need to perform, your tool should easily do it for you.

Integrated development environments (IDEs) literally provide an entire environment for your work. They bring together many different tools in a coherent way, so that the services and actions you need are seamlessly integrated together.

Some technical benefits of IDEs include the following:

- Graphical user interface (GUI) for performing actions
- Grouping of source code and configuration files into the concept of a *project*
- Tight integration with the compiler
- Coupling with a source code repository

- Ability to performance tune, analyze, and load test code
- Integration with reusable test frameworks
- Capability to utilize third-party plug-ins and tools
- Ability to debug code by executing one line at a time
- Quick access to and ease of generating project documentation

Some of the more tangible business benefits of using an IDE include the following:

- Reduces the cycle time of development
- Increases the quality and reliability of your code
- Standardizes your software development processes
- Provides a common platform for programming staff to reduce training time

Some of these benefits are definitely arguable and can sometimes be realized only after careful analysis, implementation, and execution. Many other factors come into play, but a really good Java IDE tool can be the foundation for accomplishing important milestones such as the examples I provided.

NetBeans vs. Other IDE Tools

NetBeans is my Java IDE of choice. This might be obvious since I wrote this book, but I have many valid reasons for loving and using NetBeans. My experience with development tools covers a wide range of products, such as Notepad, TextPad, Emacs, vi, Macromedia Ultradeveloper, Macromedia Dreamweaver, Oracle JDeveloper, IntelliJ IDEA, Borland JBuilder, Microsoft Visual Studio, and Eclipse.

Each of these tools has its own pros and cons. They all have devoted users and entire communities centered around them. After a while, distinguishing between the tools can be difficult, since they offer many similar features. I was on the fence deciding between IntelliJ IDEA and Eclipse. After only a few hours of working with NetBeans and viewing various tutorials, I was convinced. I downloaded, installed, and started working with it. I quickly discovered that the features were located in places I expected them to be, they functioned as I thought they would, and there were few or no configuration issues. In my opinion, that is how a tool should function out of the box.

In no particular order, the top ten reasons I think programmers should use NetBeans over another Java IDE are summarized as follows:

Intuitive and easy-to-use Matisse GUI designer for Swing development: With little or no Swing knowledge, users can be up and running dragging-and-dropping elements into a WYSIWYG design window. The Matisse GUI designer actually generates real Swing code and not the usual boilerplate fluff code many tools tend to create. At the last JavaOne conference I attended, I sat next to a gentleman who used the GUI design capabilities of JBuilder. After only two minutes of watching me use Matisse, he was completely blown away and ran off to download it for himself.

Strong refactoring support: This is particularly true for the Jackpot engine, allowing for Java type-aware refactoring using a regular expression-like query language. Designed by James Gosling, the query language is quite simple to use and allows for pattern matching and replacement. The interesting aspect to the queries is that they can be tested to match specific Java types or instances of objects.

One of the best code profilers: Given I haven't used every code profiler out there, but with an amazing array of options, the NetBeans Profiler is among the best. Users can profile for memory, CPU, and performance problems, as well as monitor threads. The Profiler can also be attached and detached from a currently running process or application. It provides 32-bit and 64-bit support, as well as allowing you to profile Enterprise JavaBeans (EJB) modules and enterprise applications. For those Mac fans in the crowd, it also supports profiling on Mac OS X Intel systems.

UML project support: Programmers can create a Unified Modeling Language (UML) project for modeling code, process steps, or design patterns. UML projects can be linked directly to Java projects. As a user creates and modifies the UML objects and diagrams, the corresponding Java code is automatically generated. If the source code in the linked Java project is changed, the diagram is also updated automatically. With the ability to export diagrams, generate code, and create web-based project reports, the UML project feature is one of the coolest additions to NetBeans that I have enjoyed using.

Ant integration: Java projects in NetBeans are structured using Ant build files. When a project is first created, the IDE generates the build script and associated targets. Users can then trigger specific targets or completely customize the structure of their build file to suit the needs of their project. For users that are not familiar with Ant, there is almost no impact, since execution of Ant targets is linked directly to the menus and buttons in NetBeans. Many users will also find it easy to import existing build files from external projects and quickly get up to speed. For beginners, it is ridiculously easy to use. For experts, it is ridiculously easy to customize.

J2ME mobile application support: Even if you don't do much mobile application development, after viewing the samples and reading an introductory tutorial, you should quickly see the power of NetBeans mobile tools. The sample applications provided are impressive enough as it is. With support for Java 2 Micro Edition (J2ME) Mobile Information Device Profile (MIDP) 2.0, a visual mobile designer, a wireless connection wizard, and over-the-air download testing, mobile application developers have some impressive and powerful tools.

Developer collaboration tools: Developers can log in to a public or private environment and share code. You can join public conversations or start your own restricted private ones. One of the greatest features I've seen in a while is the ability to drag-and-drop code or entire projects in the chat window and share code between one or more programmers. NetBeans supports multiuser team coding. As one user starts to change a block of code, it is highlighted and locked for the other users sharing it. In the current global economy, where development teams are spread across numerous locations, this tool can prove very beneficial.

Easy-to-use Update Center: The NetBeans Update Center allows you to quickly select which update distribution sites you wish to check for changes, updates, and new modules. You can also choose to install modules that you previously downloaded, but chose not to install. The Update Center is more intuitive than many other Java IDE update tools and makes updating NetBeans a snap.

Out-of-the-box JSP and Tomcat support: NetBeans comes bundled with Apache Tomcat. Once you have used the new project wizard to create a web application project, you can create your JavaServer Pages (JSP) files. Then you can right-click any JSP file and select Run File. The bundled Tomcat server immediately starts, your default Internet browser opens, and the JSP file executing in Tomcat is displayed. NetBeans is even smart enough to activate the HTTP Monitor.

NetBeans HTTP Monitor: I do a lot of web-related Java development. To me, this is one of the coolest and most unique features of any Java IDE on the market. The HTTP Monitor can be activated during the debugging or execution of a web application. It allows you to monitor the request, HTTP headers, cookies, session, servlet context, and client/server parameters. You no longer need to write server-side code to read these variables, output them to a log, and view the log file. Inside NetBeans, you can debug your code, step line by line through it, and watch the attributes you need.

These features are only a sampling of some of what NetBeans has to offer. Other Java IDEs may provide some of the capabilities described here, but none can match the NetBeans IDE's intuitive interface and integrated tool set. To learn about everything NetBeans has to offer, I invite you to continue reading the rest of the chapters in this book.

Summary

In this chapter, we discussed the high-level evolution of programming using text editors and tools. Obviously, many tools have been introduced and many milestones have been reached during the past 50 years of software development, but I wanted to cover the basic reasons why developers use full-blown IDEs.

There are many benefits to using a modern development environment like NetBeans. It offers more productivity enhancements and conveniences than competing tools, such as Notepad, TextPad, Emacs, and vi. Users can take advantage of features like an advanced GUI designer, integrated testing with JUnit, performance profiling, debugging tools, and full compilation and packaging support.

Many of the NetBeans features briefly covered in this chapter offer significant improvements over other Java IDEs. For these reasons, and many more, I believe NetBeans should be the Java IDE of choice for development teams.