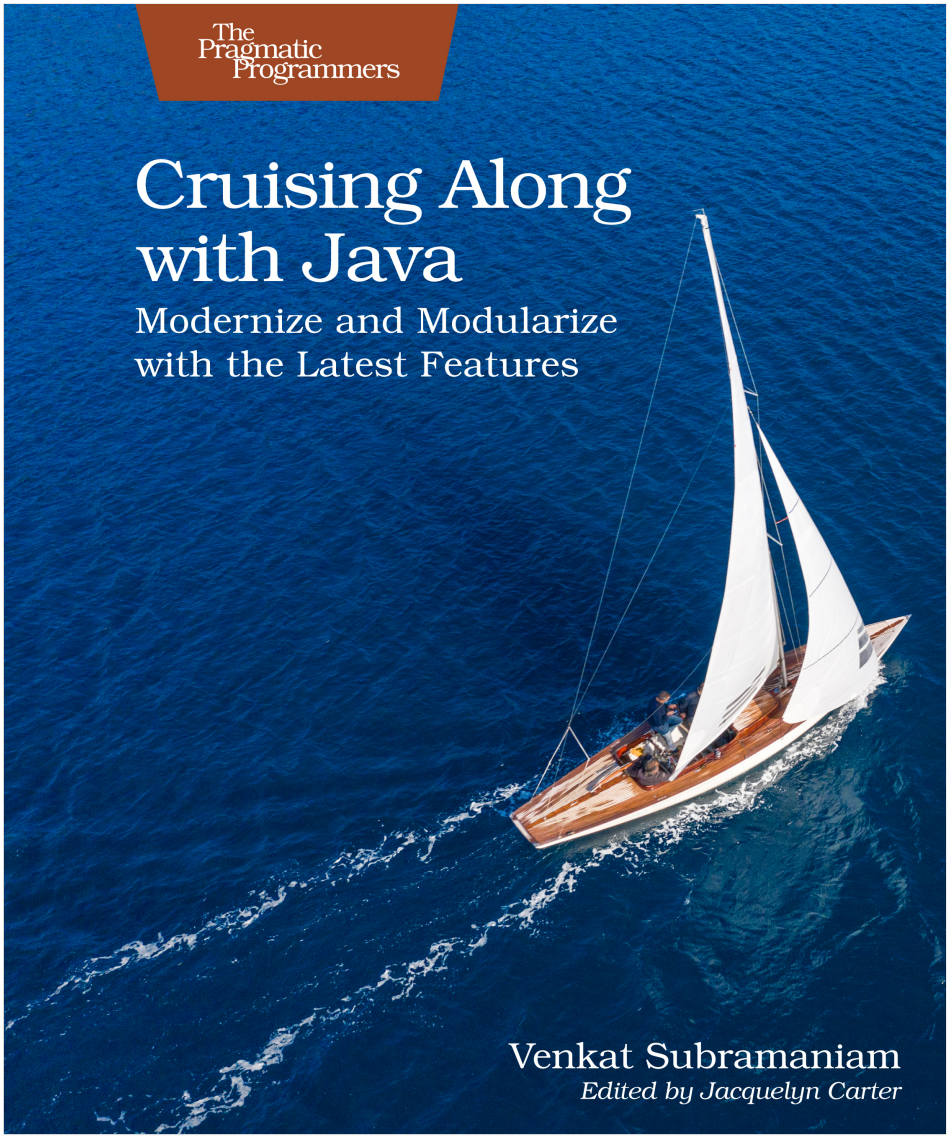


This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

Java is evolving fast. There is a new release of the language every six months. Ever since the introduction of the functional programming capabilities in Java 8, countless new features have been added. This book walks you through the most significant language changes after Java 8, from Java 9 through Java 24.

Java started out as an object-oriented programming language mixed with the imperative style of programming. Then the functional programming capabilities were added. Many developers have embraced the hybrid capabilities of the language to write code using a combination of the imperative style, the functional style, and the object-oriented paradigm.

Even though Java has pretty good OOP and functional programming support, the folks behind the language haven't been complacent. They've invested enormous time and effort to keep the language contemporary. But they don't achieve that by adding a random set of features into the language based on impulse, market pressure, or infatuation. They've been thorough in evaluating and setting the direction for the language. When considering features to add, they ask three significant questions:

- Is a feature useful for Java programmers creating enterprise and complex applications?
- Is it feasible to implement a feature without significantly compromising backward compatibility?
- Is it possible to implement a feature in a way that doesn't hinder future enhancements?

We see the results of those efforts in the steady improvements to the language over the past few years.

Java Is Agile

It's truly refreshing to see agile development in action instead of just hearing people talk about it. When the Java team announced the newer versions of the language would be released every six months—in March and September of each year—it was received with a huge amount of skepticism. Change is hard in spite of how much better the results might be. But that release cycle has been one of the most significant and bold decisions to which the team has stayed committed and on track.

In the past, the team wanted to release a new version of Java every two years. They would announce plans for what would be in the release. Developers of the language would put their sincere and hard efforts behind those planned

features. And, when the time came to release, it wasn't uncommon to hear what each one of us had said many times to our managers: "we're almost done." That "almost done" generally means "a few years later" in human timeline terms. To finalize the plan before we know the details is waterfall-like, which is how things were done before Java 9.

Agile development is *feedback-driven development* and, in essence, is guided by *adaptive planning*.

That's exactly what Java development is now.

Fast-Paced Change

Java is being released every six months, but Java is *not* being developed on a six-months timeline. It's naive to think that most complex features that have a huge impact on well over ten million developers and tens of thousands of enterprises can be developed from start to finish in six months. One of the biggest innovations behind Java is the realization that the timelines for different features don't have to be tied together into an arbitrary release.

There's a release train departing every six months. A feature can get on any release as soon as it's ready. What's in a release isn't set in stone. The plan is flexible and based on reality. The details of the features are also not committed in one shot. The features are released in preview mode and then altered based on feedback from the community at large.

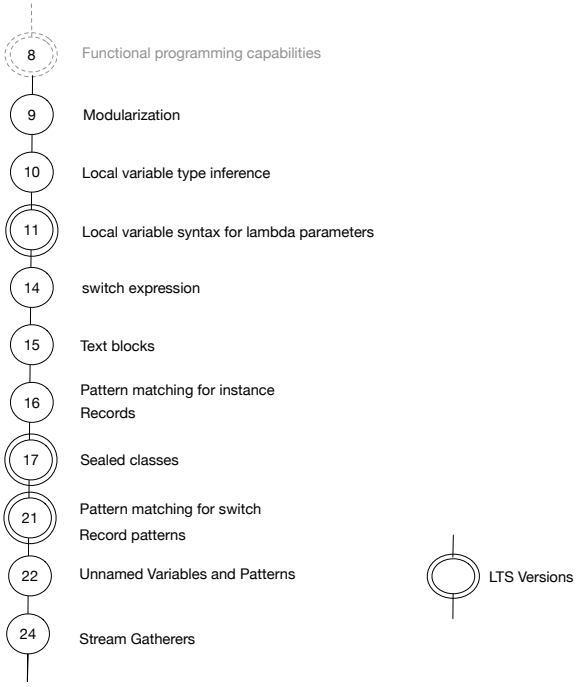
The frequent release cycles benefit the team behind the language, the developers, and the companies who make use of Java.

The developers behind the Java language are able to innovate at a faster rate thanks to the frequent release cycle. They're able to release in increments, get feedback, make changes, see their work being actively used, and move forward to build newer features. They say there's nothing more motivating than seeing their hard work benefiting the users right away.

For the users of the language, the enterprises, and the programmers building their applications, the changes now come in bite-size. They're able to use newer features much sooner rather than waiting to receive a large release once every five or so years. It's easier and more efficient to learn and use one feature every six months than six features every five years. Unlike in the past, Java developers don't feel like they're left behind on the language innovation curve working with a stagnant language. They're developing on a powerful and at the same time vibrant platform.

Recent Changes to Java

Adaptive planning and feedback driven, that's what Java is today and it's rocking. Here are the recent and exciting additions to the language—the features you'll learn about in this book—corresponding to the Java versions[1] they were finalized in.



Java 8, 11, 17, and 21 are designated as Long Term Support (LTS)[2] releases. Oracle provides premier support and periodic updates to customers who are using an LTS version. The original plan was to designate a release every three years as LTS, but that plan changed to making an LTS release every two years. Even though not all the releases of Java are LTS releases, every single one of the is developed and released with equal quality and attention to detail.

Most of the features you see in the previous figure were developed as part of incubator projects, like Project Amber,[3] which were used to explore and experiment with the design and implementation of ideas that were proposed as part of the JDK Enhancement-Proposal & Roadmap Process (JEP). Once a feature is introduced into Java, it goes through at least two rounds of preview

1. https://en.wikipedia.org/wiki/Java_version_history
2. https://www.oracle.com/java/technologies/java-se-support-roadmap.html
3. https://openjdk.org/projects/amber/

before it's accepted as an official part of the platform. This is an amazing display of *standardization after innovation*.

Moving Ahead from an LTS

The ability to upgrade every six months is superb, but that doesn't automatically result in frequent and continuous upgrades for a vast number of companies. Different organizations are at different stages of adoption of newer versions of Java. The lag is often no fault of the developers but the result of various constraints. For instance, the dependencies on third-party libraries and frameworks sometimes place limitations on upgrading. Also, the environments where their applications are deployed may place some restrictions. Some enterprises also place strict restrictions on upgrading past the versions of Java that are designated as LTS so they can reliably receive security and other updates periodically.

Depending on the company and the products that you're working on, you may have experience with one of the LTS versions and may be eager to move forward from it. This book was created for you, to take you from where you're comfortable and most experienced to where you can tap into the full potential of the language as your applications journey along the newer versions of Java.