Conclusion

I should like to end this book on a personal note. I have long since lost count of the technologies that I have seen, over four decades, announced as the future of the industry, only to be forgotten within a year or two. A much smaller number have been slow burners: technologies that have grown a faithful following without ever reaching the commercial mainstream. In this category, one that stands out is functional programming, which has attracted some of the best minds and produced some of the best ideas in software development while remaining a minority interest.

For myself, although I toyed with functional programming in the 1980s, I did not become a functional programmer. Instead, I followed the industry mainstream towards object-oriented programming, and eventually to Java. Nearly 20 years later, Java is, by most measures, the most popular programming language in the world. But, during the last few years, I have not felt complacent about the good fortune of my choice; a programmer who knows only Java would have been missing out on many useful new programming techniques that were appearing in rival languages. Many of these—for example, lazy evaluation, closures, and pattern matching—have their origins in functional languages. And this trend continues: functional programmers are optimistic about their future, above all because trends in hardware manufacturing technology and costs mean that massively concurrent systems are the future. Data immutability will be the key to reasoning about such systems.

Java is not about to become a functional language, but Java programmers should be able to take advantage of some of the insights that functional programming has developed. The changes of Java 8 are a first step in that direction. They bring immutability and lazy evaluation into practical Java programming and so address part of the great and ongoing challenge of partitioning tasks over multiple processors. Despite the commitment to backward compatibility that makes any change to a 20-year-old language so difficult, the Java design team has shown impressive ingenuity in integrating the

ideas of functional programming into a language designed on very different principles. They have made a great success of this, the biggest single set of changes in Java's history.

I am excited about these changes, and I hope to have conveyed some of this excitement to you. I hope this book has helped you to understand Java's new direction and, looking further, to become engaged with the future of the language. For the present, I would say this: programming should always be enjoyable, but you will find it much more enjoyable when you are writing the concise, readable, and performant code that Java 8 supports. I would be delighted to think that this book had contributed to making that change in your programming life.

Maurice Naftalin
Pune, August 2014