Afterword

Let's go back to high school for a moment. In algebra class, the instructor began by teaching many different manipulations such as "add the same value to both sides of the equation" or "the commutative property of addition allows us to swap its operands." This went on for several weeks, at which point the teacher switches gears and hands you a paragraph that starts off, "A train leaves New York heading West.…"After the initial panic (which you may still feel the effects of as you read the previous sentence) you settle down and express the problem in terms of an algebraic equation. You then apply the rules of algebra to the equation to arrive at an answer.

Design patterns are the word problems of the programming world; refactoring is its algebra. After having read the GoF book, you reach a point where you say to yourself, "If I had only known this pattern, my system would be so much cleaner today." The book you are holding introduces you to several sample problems, with solutions expressed in the operations of refactoring. Many people will read this book and try to memorize the steps to implement these patterns. Others will read this book and clamor for these larger refactorings to be added to existing programming tools. Both of these approaches are misguided. The true value of this book is not in the actual steps to achieve a particular pattern, but in understanding the thought processes that lead to those steps. By learning to think in the algebra of refactoring, you learn to solve design problems in behavior-preserving steps, and you are not bound by the small subset of actual problems that this book represents.

So take these exemplars that Josh has laid out for you. Study them. Find the underlying patterns of refactoring that are occurring. Seek the insights that led to the particular steps. Don't use this as a reference book, but as a primer.

John Brant
Don Roberts