Design Patterns is a modern classic in the literature of object-oriented development, offering timeless and elegant solutions to common problems in software design. It describes patterns for managing object creation, composing objects into larger structures, and coordinating control flow between objects. The book provides numerous examples where using composition rather than inheritance can improve the reusability and flexibility of code. Note, though, that it's not a tutorial but a catalog that you can use to find an object-oriented design pattern that's appropriate for the needs of your particular application—a selection for virtuoso programmers who appreciate (or require) consistent, well-engineered object-oriented designs.
Personal Review: Design Patterns: Elements of Reusable Object-Oriented Software by John M. Vlissides

This is one of the small selection of books that belongs within arm's reach of any OO developer's desk. Example code is written in C++, but the patterns are applicable to any fully-featured object-oriented language, which includes C++, Java, Smalltalk and C#, just to name a few salient examples. Itcatalogues the 23 now-classic solutions to object-oriented software design problems whose utility and generality have proven them worthy of a pattern name.

What may be less widely appreciated in the software world is that this book has influenced a more general trend towards pattern languages. This may illustrate better than anything else a central truth about the book - that it catalogues and describes existing best practice. If you have done OO design work for any length of time, you will probably have come up with a few of the patterns spontaneously, as they represent the solutions to problems that tend to arise routinely during the designer's work. This is not to say that you don't need the book - quite the opposite. Firstly, there are 23 of these patterns, and it is a designer of rare talent who has spontaneously come up with all 23. Secondly, this is rather a finished documentation of rather finished solutions, and its version of any given solution will help you avoid many pitfalls. Thirdly, we all need reference material from time to time, and this is a central foundational reference work of object-oriented design.

The examples are catalogued in a consistent and clear fashion beginning with "Intent" - why you would need it - and "Also Known As", since these patterns had already acquired names in the wild and this book seeks to organise them into a common language rather than to generate new material. "Motivation" describes the purpose of the solution in more detail, sometimes with a concrete example documented in UML and sometimes with diagrammatic portrayals of GUI goals and the like. Next comes "Applicability", which defines in bullet-point list form the circumstances in which you might want to use the pattern. "Structure" is documented in UML, now the industry standard diagramming language for object-oriented structure. "Participants" describes the object types, or classes, which go to make up the pattern. "Collaborations" describes in text form how the various participants work together and for what guarantees they rely on one another. "Consequences" describes the advantages and penalties for the design of using the pattern in question - yes, there are usually some penalties! Implementation is usually the most substantial sub-section and includes an example implementation programmed in C++ and detailed discussion of the concrete design decisions. "Known Uses" documents some examples of the use of the pattern that have been captured in the wild, while "Related Patterns" closes each section by documenting other patterns which are either interchangeable or interoperable with the pattern.

The patterns themselves are grouped into Creational, Structural and Behavioural categories. Usefully, the inside covers are printed with an
index of the patterns in the three groups described above, and at the back a Relationship Diagram showing the links between the different patterns and examples of the graphical notation indicating how Class Diagrams, Object Diagrams and Interaction Diagrams are laid out. All three of these can be studied in more depth in a UML primer such as UML Distilled: A Brief Guide to the Standard Object Modelling Language (3rd Edition).

All-in-all, the book is a masterpiece of clarity in software literature and a true classic, if a little dry. Most of us use a few of the patterns routinely, even unthinkingly, a few occasionally, and a bunch of them not at all. I encountered the Visitor pattern while working for an Investment Banking team and found I hardly knew it, while I used the Decorator Pattern almost to excess in my own HeadCase application. Other books actually warn against "Singletonitis", so overused has the pattern become. This volume is agnostic, merely documenting how to do it and what the pros and cons are. It is also a standard work, so an OO designer can be expected to be familiar with at least some of it at interview. It would seem unwise to do without a copy ready to hand.

Correction: As a commentator has pointed out, technically the diagrams are in OMT, having been published prior to the standardisation of UML. (Which is still growing to this day.) As OMT is succeeded and subsumed by UML, strictly the diagrams are both OMT and UML, but the academic detail deserves to stand corrected, even if it would be of little interest to any serious professional using the book for practical reference. At any rate, all the diagram types used can be found in any standard work on UML.

For More 5 Star Customer Reviews and Lowest Price:
Design Patterns: Elements of Reusable Object-Oriented Software by John M. Vlissides 5 Star Customer Reviews and Lowest Price!