Introduction to Software Patterns

Software patterns are reusable solutions to recurring problems that occur during software development. Because this book is all about software patterns, they are simply referred to in the rest of this book as patterns.

What makes a bright experienced programmer much more productive than a bright but inexperienced programmer is experience. Experience gives programmers a variety of wisdom. As programmers gain experience, they recognize some new problems as being similar to problems that they have solved before. As they gain even more experience, they recognize that the solutions that they use to solve these similar problems follow patterns that repeat themselves. By being aware of these patterns, experienced programmers are able to recognize situations that patterns apply to and immediately use the solution without having to stop, analyze the problem and pose possible strategies.

When a programmer discovers a pattern, it is just an insight. In most cases, to go from an unverbalized insight to a well thought out idea that the programmer can clearly articulate is surprisingly difficult. It is also an extremely valuable step. When we understand a pattern well enough to put it into words, we are able to intelligently combine it with other patterns. More importantly, once put into words, a pattern can be used in discussions among programmers who know the pattern. That allows programmers to more effectively collaborate and combine their wisdom. It can also help avoid the situation of programmers arguing over different solutions to a problem, only to find that they were really thinking of the same solution but expressing it in different ways.

Putting a pattern into words has an additional benefit for less experienced programmers who have not yet discovered the pattern. Once a pattern has been put into words, more experienced programmers can teach it to programmers to do not know the pattern.

The value of this book is that it will give experienced programmers a common vocabulary to discuss patterns. It also allows programmers who have not yet discovered a pattern to learn about the pattern.

Though this book includes a substantial breadth of patterns, there are additional patterns that the author knows but did not have time to put in the book. You, dear reader, may discover some of those yourself. Some patterns you discover may be highly specialized and only of interest to a small number of people. Other patterns may be of very broad interest and worthy of inclusion in this book.

The patterns cataloged in this book convey constructive ways of organizing parts of the software development cycle. There are other patterns that recur in programs that are not constructive. These are referred to as anti-patterns, because anti-patterns can cancel out the benefits of patterns. This book does not attempt to catalog anti-patterns.

Description of Patterns

Patterns are usually described using a format that includes the following information:

- A description of the problem that includes a concrete example and a solution specific to the concrete problem.
- A summary of the considerations that lead to the formulation of a general solution.
- A general solution.
- The Consequences, good and bad, of using the given solution to solve a problem.
- A list of related patterns.

The details of how that information is presented are different in different books. The format used in this set of books varies with the phase of the software life cycle that the pattern addresses. The patterns in this volume are all related to the design phase of the software life cycle. The descriptions of design phase related patterns in the volume are organized with the following headings into sections.

Pattern Name

The heading of this section consists of the name of the pattern and a bibliography reference indicating where the pattern came from. Most patterns do not have any additional text under this heading. For those that do, this section contains information about the derivation or general nature of the pattern.

Synopsis

This section contains a brief description of the pattern that is no longer than a few sentences. The synopsis conveys the essence of the solution provided by the pattern. The Synopsis is primarily directed at experienced programmers who may recognize the pattern as one they already knew, but may not have had a name for. After recognizing the pattern from its name and synopsis, it may be sufficient to skim over the rest of the pattern description.

Programmers who do not recognize a pattern from its name and synopsis, should not be discouraged. Instead, they should carefully read through the rest of the pattern description carefully to understand it.

Context

This section describes the problem that the pattern addresses. For most patterns, the problem is presented in terms of a concrete example. After presenting a concrete problem, the context section presents a design solution to the concrete problem.

Forces

The Forces section summarizes the considerations that lead to the general solution to the problem presented in the following section.

Solution

The Solution section is the core of the pattern. It describes a general purpose solution to the problem that the pattern addresses.

Consequences

The consequences section explains the implications, good and bad, of using the solution presented in the previous section.

Implementation

The implementation section describes important considerations to be aware of when implementing the solution. It also may describe some common variations or simplifications of the solution.

Java API Usage

Where there is a suitable example of the pattern in the core Java API, it is pointed out in this section. Patterns that are not used in the core Java API do not have this section in their descripion.

Code Example

This section contains a code example that shows a sample implementation of a design that uses the pattern. Usually, the design that is implemented in this section is the design described previously in the "Context" section.

Related Patterns

This section contains list of pattern that are related to the pattern being described.

A Brief History of Patterns

The idea of software patterns originally came from the field of architecture. An architect named Christopher Alexander wrote some books describing patterns in building architecture and urban planning. Some of those book are

A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977
The Timeless Way of Building, Oxford University Press, 1979

The ideas presented in those books are applicable to a number of fields outside of architecture, including software.

In 1987, Ward Cunningham and Kent Beck used some of Alexander's ideas to develop five patterns to guide the design of user interfaces. They published a paper on them at OOPSLA-87. The paper was titled, "Using Pattern Languages for Object-Oriented Programs".

In the early 1990s, four authors begin work in a very influential book called Design Patterns. The four authors were ErichGamma, RichardHelm, JohnVlissides and RalphJohnson. Their book was published in 1994. It popularized the idea of patterns and was the largest single influence on this book. The Design Patterns book is often called the "gang of four book" or GoF.

This book represents an evolution of Patterns and objects since the GoF book was published. The GoF book used C++ for its examples. This book uses Java and takes a rather Java-centric view of most things. When the GoF book was written, UML did not exist. It is now widely accepted as the preferred notation for object oriented analysis and design, so that is the notation used in this book.

Finally, this set of books covers a wider range of patterns than previously published works. Even with the goal of providing coverage of a broad selection of patterns, time constraints have limited the number of patterns that can be included. Two volumes of this work are currently planned. If there is sufficient interest, more volumes will follow.