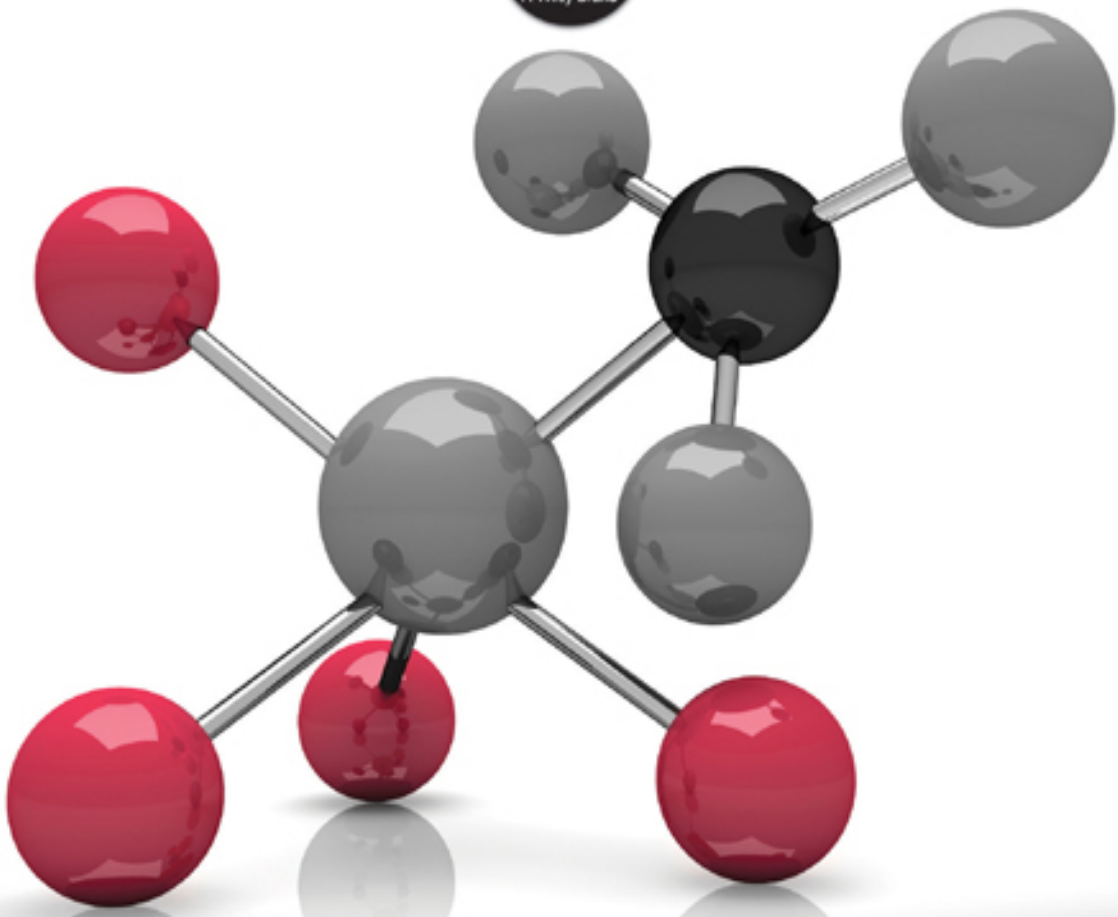
Join the discussion @ p2p.wrox.com

Wrox Programmer to Programmer™



Professional Java® EE Design Patterns

Murat Yener, Alex Theedom

CONTENTS

List of Tables

List of Illustrations

FOREWORD

> *Ignorant men raise questions that wise men answered a thousand years ago*
>
> —JOHANN WOLFGANG VON GOETHE

Design patterns are our link to the past and the future. They make up a foundational language that represents well understood solutions to common problems that talented engineers before us have added to our collective knowledge base. Design patterns or blueprints exist in every engineering field in one way or another. Software development is no different. Indeed, design patterns are probably our most tangible link to engineering rather than the more organic and less regimented world of the artisan or craftsman. The art and science of design patterns was brought to the world of software engineering—and more specifically to enterprise Java—by the seminal Gang of Four (GoF) book. They have been with us ever since through our adventures in J2EE, Spring, and now modern lightweight Java EE. This is for very good reasons. Server-side Java developers tend to write the type of mission critical applications that need to stand the test of time and hence benefit the most from the discipline that design patterns represent.

It really takes a special kind of person to write a book on design patterns, let alone a book on how to utilize design patterns in Java EE applications. You require not only basic knowledge of APIs and the patterns themselves, but deep insight that can only come with hard-earned experience, as well as an innate ability to explain complex concepts

elegantly. I am glad Java EE now has Murat and Alex to accomplish the mighty feat.

This book fulfills a much needed gap and fills it well. It is also very good that the book is on the cutting edge and covers Java EE 7 and not just Java EE 6 or Java EE 5. In fact many of the design patterns covered, like Singleton, Factory, Model-View-Controller (MVC), Decorator, and Observer, are now incorporated right into the Java EE platform. Others like Facade, Data Access Object (DAO), and Data Transfer Object (DTO) fit elegantly on top. Murat and Alex tackle each pattern, explain its pragmatic motivation, and discuss how it fits into Java EE.

It is an honor and a privilege to write a small opening part of this very important book that I hope will become a very useful part of every good Java EE developer's bookshelf. I hope you enjoy the book, and that it helps you write better, more satisfying enterprise Java applications.

M. REZA RAHMAN
Java EE/GlassFish Evangelist
Oracle Corporation

INTRODUCTION

THIS BOOK DISCUSSES THE CLASSIC DESIGN PATTERNS that were first mentioned in the famous book by the GoF[1] and updates them specifically for Java EE 6 and 7.

In every chapter we describe the traditional implementation of each pattern and then show how to implement it using Java EE-specific semantics.

We use full code examples to demonstrate both the traditional and Java EE implementations and color each chapter with real-life stories that show the use (or misuse) of the pattern.

We investigate the pros and cons of each pattern and examine their usages. Each chapter finishes with some exercises that challenge your understanding of the pattern in Java EE.

WHO THIS BOOK IS FOR

This book is for everyone with any level of experience. It covers almost everything about a pattern, from how it is referred to in other books, to code on basic Java implementation, to Java EE implementation, and finally real life examples of how and when to use a specific pattern. It also has real life war stories that talk about good and bad practices.

Having some basic knowledge of design patterns and Java EE will aid you as you read this book.

If you are already experienced with patterns and basic Java implementations, you may prefer to jump into Java EE implementations. Refreshing your memory and knowledge of design patterns could prove helpful.

WHAT THIS BOOK COVERS

This book covers all classical design patterns that Java EE offers as part of standard implementation, besides some new patterns. The coverage goes back to Java EE5 and is up to date for the latest version available, which is Java EE 7.

We hope this book will be a reference you will keep on your shelf for a long time.

HOW THIS BOOK IS STRUCTURED

Each chapter focuses on a design pattern. If the pattern is classical, a simple Java implementation is given after the explanation of the pattern. Each chapter offers war stories telling a good or bad real life example about the pattern focused on/in the chapter. The war story is followed by a Java EE implementation, example, and explanation. Each code sample given can be run by itself. Finally, each chapter ends with when and how to use the pattern effectively.

WHAT YOU NEED TO USE THIS BOOK

Any modern computer with an operating system that has a Java Virtual Machine (JVM) implementation is sufficient to run the samples given in this book. For ease of coding, you need an integrated development environment (IDE) of your own choice. The sample can run on any popular modern IDEs including Eclipse, NetBeans, and IntelliJ.

You need the Java Development Kit (JDK) for Java EE7 to be able to compile and run the code samples, but some of the code samples would also work on previous Java EE JDKs.

You can use any Java EE7–compliant application server to run the samples. We ran all the code samples on Glassfish, which is the reference implementation server, and TomEE, which is the Java EE version of the popular Java web server Tomcat. You can use any server, but because Glassfish is the reference implementation, you might want to try it for the samples.

To run the samples in this book, you need the following:

- An operating system that has a JDK for Java EE7, such as Linux, Mac OS X, or Windows

- Java EE 7 JDK
- An IDE of your choice, such as Eclipse for Java EE Developers, NetBeans, or IntelliJ
- Java EE 7–compliant application server such as GlassFish or TomEE

The source code for the samples is available for download from the Wrox website at:

www.wrox.com/go/projavaeedesignpatterns

MOTIVATION FOR WRITING

In November 2011, after having a debate on Java EE versus Spring for a project, I went back to my desk and wrote a blog post titled "Java EE 6 and the Ewoks,"[2] which became popular pretty quickly. The story was based on the TV show *How I Met Your Mother*. In this show, Barney, who is the playboy character, introduced a theory that was focused on Ewoks, the teddy bear–like creatures introduced in Episode VI of *Star Wars*. Fans have mixed feelings on Ewoks.

According to Barney, those born before May 25, 1973, when *Return of the Jedi* was released, think Ewoks are childish and simply hate them. However, those born after that date find Ewoks cute because they remind them of teddy bears.

Now back to my story. Engaging in a debate with a customer about Java EE versus Spring made me realize that it's similar to the Ewok theory. Those who are old enough to have used J2EE 1.4 (EJB 1.0/2.0/2.1) in corporate projects had a slow, unproductive development environment with RAM-eating and buggy IDEs and servers taking several minutes to boot. The architecture was over engineered and probably failed, resulting in a migration to

Spring. Those users tended to hate Java EE with a passion, no matter what version they used. The release of Java EE 5 was underrated and did not really impress anyone.

Java EE will never be J2EE again. It is now open, has a large community and reshapes itself by assimilating good ideas from frameworks such as Spring and Hibernate. The first great change was the architecture and style of coding. Enterprise JavaBeans (EJB) followed the lightweight Plain Old Java Object (POJO) model, almost unusable entity beans were replaced with Java Persistence API (JPA), REST and Web Services became standard and integral parts of the run time, and annotations replaced XML configuration. Still, some might argue that Java EE 5 was not ready for the huge shift because it was not as mature as Spring, and the development environment was still not responsive enough. Using Spring on Tomcat instead of EJBs and Java EE 5 on an application server greatly increased the development productivity, but Java EE 5 was still a big step forward towards designing, leveraging, and architecting the Enterprise Java platform from scratch.

This shift was followed by Java EE 6 and 7, which used the same principles and ideas as Java EE 5. Java EE is a great choice for development, but the debate was not over, thanks to the Ewok theory.

It was a hot August day when I first got a call from Wrox/Wiley about whether I would be interested in writing a Spring book. I was experienced and confident with implementing and developing in Spring, but there were already tons of books written about it, which made it hard to see the value in writing a new one.

Besides, I was using Java EE more than ever since version 6 had been released. Considering the Spring versus Java EE debates, my blog posts, and the Ewoks, I felt like writing about Java EE. However, just like Spring, there

were many great Java EE books that I admired. I always had the feeling that some properties of Java EE were underrated. Java EE has great built-in implementations of design patterns with simple use of annotations.

The classic patterns listed in the GoF book were used extensively in almost all languages, frameworks, and platforms. J2EE was no exception and neither was Java EE. Actually Java EE took a bold step in providing default implementations for many of those patterns, but still even most of the experienced developers underestimated the value of those out of the box implementations.

I had been blogging about those patterns for almost a year, so I decided to present a counteroffer to write a book on the “classic” design patterns in Java EE. As you are reading this book now, you may guess the feedback was positive.

This book fills the gap between the Java EE platform with the classic design patterns from the GoF book as well as talking about new patterns. This way we did not write just another Java EE book but a catalogue for design patterns in Java EE.

I started blogging, writing and giving talks on design patterns in Java EE to extend my knowledge and experience on a platform I really believed in, so the best thing about writing this book for me was that I had the chance to write about something I was really passionate about. Although my blog had simpler examples, I was already using it as a reference when I needed, so writing a book, which is more formally and properly formatted while still following the same idea was a great opportunity.

Every chapter that my coauthor Alex and I wrote had the same goal: Write content that we would like to read ourselves. The result is a book that we both want to keep as a reference.

We hope that you enjoy reading this book as much as we enjoyed writing it.

CONVENTIONS

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

> **NOTE** *Notes indicates notes, tips, hints, tricks, or asides to the current discussion.*

As for styles in the text:

- We *highlight* new terms and important words when we introduce them.
- We show keyboard strokes like this: Ctrl+A.
- We show file names, URLs, and code within the text like so: `persistence.properties`.
- We present code in two different ways:

  We use a monofont type with no highlighting for most code examples.

  We use bold to emphasize code that is particularly important in the present context or to show changes from a previous code snippet.

SOURCE CODE

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All the source code used in this book is available for download at

www.wrox.com. Specifically for this book, the code download is on the Download Code tab at:

www.wrox.com/go/projavaeedesignpatterns

You can also search for the book at www.wrox.com by ISBN (978-1-118-84341-3) to find the code. A complete list of code downloads for all current Wrox books is available at www.wrox.com/dynamic/books/download.aspx.

Each chapter starts with introducing a basic Java implementation of the pattern, if there is any. Next, the chapter lists a Java EE implementation of the pattern that can only compile and run on the Java EE JDK and a Java EE–compliant application server.

Most of the code on www.wrox.com is compressed in .ZIP, .RAR, or a similar archive format appropriate to the platform. Once you download the code, just decompress it with an appropriate compression tool.

ERRATA

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or a faulty piece of code, we would be very grateful for your feedback. By sending in errata, you may save another reader hours of frustration. At the same time, you will be helping us provide even higher quality information.

To find the errata page for this book, go to:

www.wrox.com/go/projavaeedesignpatterns

Then click the Errata link. On this page, you can view all errata that has been submitted for this book and posted by Wrox editors.

If you don’t spot “your” error on the Book Errata page, go to www.wrox.com/contact/techsupport.shtml and complete the form there to send us the error you have found. We’ll check the information and, if appropriate, post a message to the book’s errata page and fix the problem in subsequent editions of the book.

P2P.WROX.COM

For author and peer discussion, join the P2P forums at http://p2p.wrox.com. The forums are a web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At http://p2p.wrox.com, you will find a number of different forums that will help you, not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to http://p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Complete the required information to join, as well as any optional information you wish to provide, and click Submit.
4. You will receive an e-mail with information describing how to verify your account and complete the joining process.

NOTE *You can read messages in the forums without joining P2P, but in order to post your own messages, you must join.*

Once you join, you can post new messages and respond to messages that other users post. You can read messages at any time on the web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to This Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works, as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.

CONTACT THE AUTHORS

If you have any questions regarding the contents of this book, the code, or any other related matter you can contact the authors directly on their blogs and via Twitter.

Murat Yener:

- Blog—`devchronicles.com`
- Twitter—`@yenerm`

Alex Theedom:

- Blog—`alextheedom.com`
- Twitter—`@alextheedom`

NOTES

1. *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley, 1994): Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides.

2. *Java EE 6 and the Ewoks:* http://www.devchronicles.com/2011/11/javaee6-and-ewoks.html.

PART I
Introduction to Java EE Design Patterns

1
A Brief Overview of Design Patterns

WHAT'S IN THIS CHAPTER?

- An overview of design patterns
- A short history about design patterns and why they are important
- The use of design patterns in the real world
- The history and evolution of Java Enterprise Edition
- The emergence of enterprise patterns
- How these design patterns have evolved in the enterprise environment
- Why and how patterns become anti-patterns

This book is aimed at bridging the gap between the traditional implementation of design patterns in the Java SE environment and their implementation in Java EE.

If you are new to design patterns, this book will help you get up to speed quickly as each chapter introduces the design pattern in a simple-to-understand way with plenty of working code examples.

If you are already familiar with design patterns and their implementation but are not familiar with their implementation in the Java EE environment, this book is perfect for you. Each chapter bridges the gap between the traditional implementation and the new, often easier, implementation in Java EE.

If you are an expert in Java, this book will act as a solid reference to Java EE and Java SE implementations of the

most common design patterns.

This book focuses on the most common Java EE design patterns and demonstrates how they are implemented in the Java EE universe. Each chapter introduces a different pattern by explaining its purpose and discussing its use. Then it demonstrates how the pattern is implemented in Java SE and gives a detailed description of how it works. From there, the book demonstrates how the pattern is now implemented in Java EE and discusses its most common usage, its benefits, and its pitfalls. All explanations are accompanied by detailed code examples, all of which can be downloaded from the website accompanying this book. At the end of each chapter, you'll find a final discussion and summary that rounds up all you have read in the chapter. There are even some interesting and sometimes challenging exercises for you to do that will test your understanding of the patterns covered in the chapter.

WHAT IS A DESIGN PATTERN?

> *Design patterns are "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context."*
>
> —GANG OF FOUR

Design patterns offer solutions to common application design problems. In object-oriented programming, design patterns are normally targeted at solving the problems associated with object creation and interaction, rather than the large-scale problems faced by the overall software architecture. They provide generalized solutions in the form of boilerplates that can be applied to real-life problems.

Usually design patterns are visualized using a class diagram, showing the behaviors and relations between

classes. A typical class diagram looks like Figure 1.1.

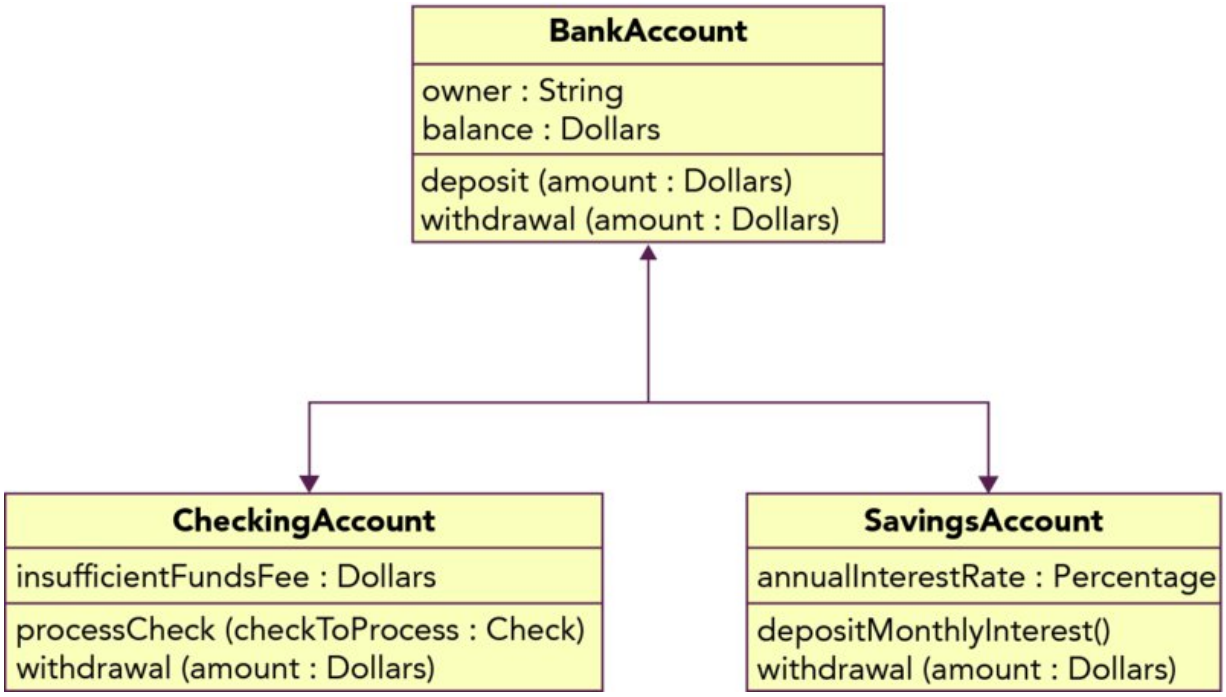


Figure 1.1 A class diagram showing inheritance

Figure 1.1 shows the inheritance relationship between three classes. The subclasses `CheckingAccount` and `SavingsAccount` inherit from their abstract parent class `BankAccount`.

Such a diagram is followed by an implementation in Java showing the simplest implementation. An example of the singleton pattern, which will be described in later chapters, is shown in Figure 1.2.

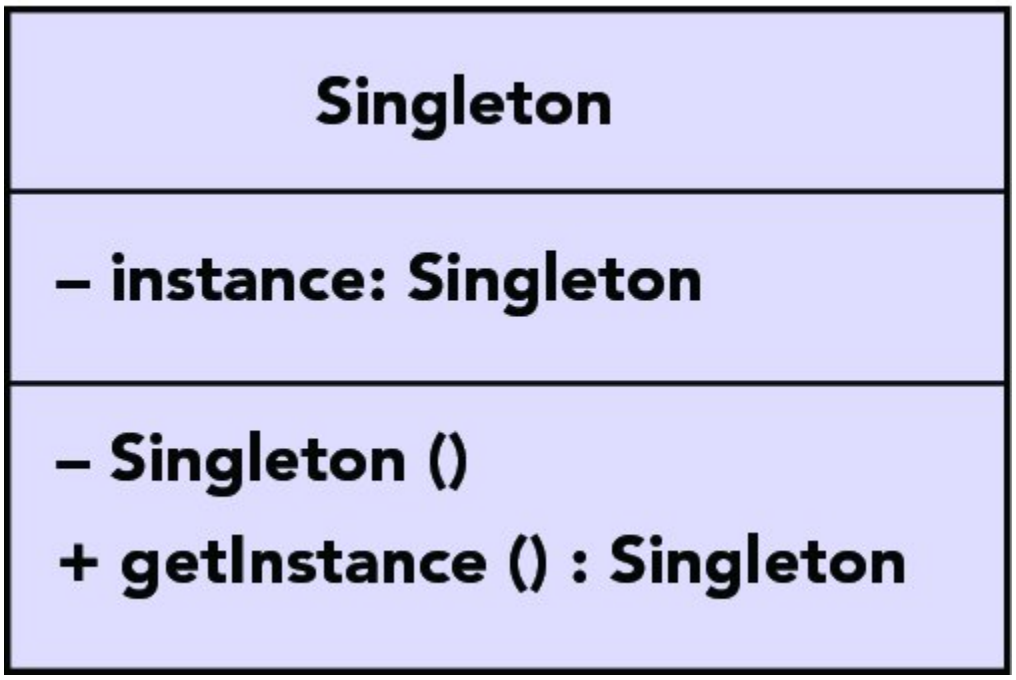


Figure 1.2 The singleton pattern class diagram

And here is an example of its simplest implementation.

```
public enum MySingletonEnum {
    INSTANCE;
    public void doSomethingInteresting(){}
}
```

How Patterns Were Discovered and Why We Need Them

Design patterns have been a hot topic since the famous Gang of Four (GoF, made up of Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) wrote the book *Design Patterns: Elements of Reusable Object-Oriented Software*,[1] finally giving developers around the world tried and tested solutions to the commonest software engineering problems. This important book describes various development techniques and their pitfalls and provides 23 object-oriented programming design patterns. These patterns are divided into three categories: creational, structural, and behavioral.

But why? Why did we suddenly realize we needed design patterns so much?

The decision was not that sudden. Object-oriented programming emerged in the 1980s, and several languages that built on this new idea shortly followed. Smalltalk, C++, and Objective C are some of the few languages that are still prevalent today. They have brought their own problems, though, and unlike the development of procedural programming, this time the shift was too fast to see what was working and what was not.

Although design patterns have solved many issues (such as spaghetti code) that software engineers have with procedural programming languages like C and COBOL, object-oriented languages have introduced their own set of issues. C++ has advanced quickly, and because of its complexity, it has driven many developers into fields of bugs such as memory leaks, poor object design, unsafe use of memory, and unmaintainable legacy code.

However, most of the problems developers have experienced have followed the same patterns, and it’s not beyond reason to suggest that someone somewhere has already solved the issues. Back when object-oriented programming emerged, it was still a pre-Internet world, and it was hard to share experiences with the masses. That’s why it took a while until the GoF formed a collection of patterns to well-known recurring problems.

Patterns in the Real World

Design patterns are infinitely useful and proven solutions to problems you will inevitably face. Not only do they impart years of collective knowledge and experience, design patterns offer a good vocabulary between developers and shine a light on many problems.

However, design patterns are not a magic wand; they do not offer an out-of-the-box implementation like a framework or a tool set. Unnecessary use of design patterns, just because they sound cool or you want to impress your boss, can result in a sophisticated and overly engineered system that doesn't solve any problems but instead introduces bugs, inefficient design, low performance, and maintenance issues. Most patterns can solve problems in design, provide reliable solutions to known problems, and allow developers to communicate in a common idiom across languages. Patterns really should only be used when problems are likely to occur.

Design patterns were originally classified into three groups:

- **Creational patterns**—Patterns that control object creation, initialization, and class selection. Singleton (Chapter 4, "Singleton Pattern") and factory (Chapter 6, "Factory Pattern") are examples from this group.
- **Behavioral patterns**—Patterns that control communication, messaging, and interaction between objects. The observer (Chapter 11, "Observer Pattern") is an example from this group.
- **Structural patterns**—Patterns that organize relationships between classes and objects, providing guidelines for combining and using related objects together to achieve desired behaviors. The decorator pattern (Chapter 7, "Decorator Pattern") is a good example of a pattern from this group.

Design patterns offer a common dictionary between developers. Developers can use them to communicate in a much simpler way without having to reinvent the wheel for every problem. Want to show your buddy how you are planning to add dynamic behavior at run time? No more