Professional Expertise Distilled

Learning Modular Java Programming

Explore the power of modular programming for building applications with Java and Spring!

Tejaswini Mandar Jog

[PACKT] enterprise
professional expertise distilled
PUBLISHING

Learning Modular Java Programming

Explore the power of modular programming for building applications with Java and Spring!

Tejaswini Mandar Jog

[PACKT] PUBLISHING enterprise
professional expertise distilled

BIRMINGHAM - MUMBAI

Learning Modular Java Programming

First published: June 2016

Production reference: 1270616

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-78588-882-3

www.packtpub.com

Credits

Author
Tejaswini Mandar Jog

Reviewer
Dionisios Petrakopoulos

Acquisition Editor
Larissa Pinto

Content Development Editor
Shali Deeraj

Technical Editor
Anushree Arun Tendulkar

Copy Editor
Safis Editing

Project Coordinator
Sanchita Mandal

Proofreader
Safis Editing

Indexer
Monica Ajmera Mehta

Graphics
Jason Monteiro

Production Coordinator
Aparna Bhagat

Cover Work
Aparna Bhagat

About the Author

Tejaswini Mandar Jog is a passionate and enthusiastic SCJP-certified trainer. She has more than eight years' experience in the IT training field, specializing in Java, J2EE, and relevant technologies. She has worked with many renowned corporate companies on training and skill enhancement programs. She is also involved in the development of projects using Java, Spring, and Hibernate.

I still remember the very first conversation with the editor about the book. Before that day I was just a reader, I never thought of writing a book. But now I had an opportunity to be an author! I was in a dilemma, and confused—can I be? But then my personal guide and philosopher Mandar gave me the confidence to go for it. Thank you Mandar not just for providing strong support, but also for your valuable suggestions, which helped me to improve the contents.

This book would not have been completed without the help of Shali, the editor. Her suggestions and efforts made the difference. Aaron, the acquisition editor, thank you for showing confidence in me, and giving me the confidence to write this book.

Without the best wishes and support of my family, it is doubtful that I would have completed this book.

It would be mean of me not to mention Ojas, my lovely son. I really appreciate the way he supported me, so that I was able to write peacefully. Love you a lot dear!!!

Finally, thank you to all who helped by supporting me directly and indirectly to complete this book. Thank you all just for being with me!!!

About the Reviewer

Dionisios Petrakopoulos has worked in several companies, using different programming languages (C, C++, Java SE, Java EE, and Scala) and technologies, as a senior software engineer for the past 15 years. His main interest is the Java ecosystem and the various facets of it. His other area of interest is information security, and especially cryptography. He holds a BSc in computer science and an MSc in information security, both from Royal Holloway, University of London.

I would like to thank my wife, Anna, for her support and love.

Table of Contents

Preface **v**

Chapter 1: Introducing Modular Programming **1**

Software – the perspective **1**

Modules **3**

What is behind and in a module? 3

The practical aspect 3

The gang – modular programming **4**

The world of modules 4

Tiers and layers in an enterprise application **5**

One-tier applications 7

Two-tier applications 7

Three-tier applications 8

N-tier applications 9

Java Enterprise architecture 11

Sharing the work **12**

Coordinate with the team **13**

Versioning tools 13

Centralized versioning 14

The architecture of SVN 15

Distributed versioning 17

Summary **17**

Chapter 2: Saying Hello to Java EE **19**

The enterprise as an application **19**

The Java EE platform **20**

Features of the Java EE platform 21

The world of dotcoms 22

Servlet – the dynamicity 23

MVC I architecture 24

MVC II architecture 25

The practical aspect 25
What is a framework? 29
Spring MVC 30
The components 31
The front controller 32
Spring MVC controller 32
ModelAndView 32
ViewResolver 33
The configuration file 33
Summary 39
Chapter 3: Implementing the Presentation Layer 41
Presentation 41
Data binding 43
Case 1 – Reading request parameters for searching 43
Case 2 – Reading multiple form fields 46
Form validation 53
Developing customized validators using Spring validators 54
Annotation-based validations 59
Summary 65
Chapter 4: Talking to the Database 67
Persistence 67
Using object serialization 68
Disadvantages of using object serialization 68
Storing data in XML 68
Disadvantages of storing data in XML 68
Saving the data in a relational database 68
Advantages of saving data in a relational database 69
Interaction of Java with relational databases 69
Types of JDBC drivers 70
JDBC-ODBC bridge driver 70
JDBC Native API Driver/Partly Java Driver 71
JDBC Net Protocol Driver 72
All Java drivers 73
Spring-JDBC integration 76
Configuring DataSource in Spring JDBC 77
Types of integration of JDBC 78
Integrating the DataSource to get a connection reference 79
Integrating the JDBC template 82
Integrating JDBC DAO support 84
Problems with JDBC 86
Introduction to ORM 86
Advantages of using ORM 86
Introduction to Hibernate 86
Hibernate architecture 87

Spring Hibernate integration 93

Introduction to unit testing 98

Unit testing using JUnit 98

Steps for writing a TestCase using annotation 98

Summary 102

Chapter 5: Developing the Business Layer 103

Business logic 104

Domain knowledge 106

Rules, formulas, and conditions 107

Case studies 107

Developing the business layer 107

Transaction management 111

JDBC and transaction management 112

Spring and transaction management 112

Programmatic transaction 113

Declarative transaction 113

Declarative transaction management 117

Programmatic transaction management 119

Summary 121

Chapter 6: Testing Your Application 123

Software testing 123

The waterfall model 124

The spiral model 125

The V model 125

Verification phases 126

Validation phases 126

Mock testing 127

Spring testing framework 127

Case1 – Inserting contact with correct values as per validation rules 128

Case2 – Inserting a contact by violating validation rules for contacts 130

Why integration testing? 133

Mockito testing 141

Arquillian 146

Summary 147

Chapter 7: Securing the Application 149

Make it safe, make it secure 149

Spring security framework 151

Secure web request 152

Way 1 – Spring Security for URL using servlet filters 153

Case 1 – Basic authentication 157

Case 2 – Login form authentication 161

Case 3 – Authentication against database 163

Case 4 – Remember me 165
Case 5 – Logout 167
Way 2 – Spring Security using AOP 168
@Secured 168
@RolesAllowed 168
SpEL-enabled security annotations for securing the methods 169
Spring Security using pointcut 171
Way 3 – Custom security 172
Summary 172
Chapter 8: Versioning and Deploying 173
Versioning 173
Collabnet server 174
Visual SVN server 180
Adding SVN as a plugin to Eclipse 188
Adding files in the project and committing them to the repository 193
Importing the project in the workspace 194
Updating and tracking the project for latest changes in the repository 196
Project deployment 198
Copying a WAR file into Tomcat without Tomcat manager 198
Copying a WAR file into Tomcat with Tomcat manager 201
Summary 203
Index 205

Preface

Welcome to the world of Java EE development! A huge world, with a large number of things to learn and so many skills to adapt. It's actually difficult to decide what to start with. When I started, I faced a similar problem. Now, also when I am in training sessions or seminars, I find many people who want be professional developers, but don't have much exposure to the processes, stages, and thinking involved in application development. This book helps you by providing a path for web development that can used to understand the process of Java modular development through an easy-to-understand case study. Nowadays, in Java EE, there are many technologies in the market. One such technology is Spring.

Spring is useful for developing independent Java modules that can then be combined to create a complete application. We have used Spring, Spring MVC, and many of its features throughout the book, while discussing the concepts of database, unit testing, security, and many other topics.

What this book covers

Chapter 1, *Introducing Modular Programming*, starts with a discussion about Enterprise application, its architecture, and its development. Enterprise application development is a team that activity faces many problems concerning collaboration between team members. We will introduce coordinated development and the tools involved in this chapter.

Chapter 2, *Saying Hello to Java EE*, involves a short warm up by discussing and developing a Java web application using Servlet-JSP. We will redevelop the application using a Spring to get startup gear as Spring MVC developer.

Chapter 3, *Implementing the Presentation Layer*, discusses the points that need to be taken care of when developing the most important layer of an application: the presentation layer. We will discuss how to develop the pages to incorporate data binding for business logic, as well as for presentation, using Spring MVC features.

Chapter 4, *Talking to the Database*, discusses Spring JDBC connectivity. Data collected from the user and data to be used in the business logic need to be persisted. We will also cover Spring DAO support persistency. We will then move on a step and introduce Hibernate, the ORM technology, and its integration with Spring. We will also cover unit testing to make sure our code is working fine.

Chapter 5, *Developing the Business Layer*, discusses the development of the most important layer of an application—the business layer—and the communication between the layers. An application needs to be developed by following a number of business rules.

Chapter 6, *Testing Your Application,* explains that the modules developed by the developer should produce the correct result. To ensure the correctness of the code in this chapter, we will cover the basics of testing with the help of the V module. We will also cover integration testing with JUnit and Mockito.

Chapter 7, *Securing the Application,* discusses why and how to secure the application. In an application, there are certain modules that are open and available to all, and some that are restricted. We will apply the Spring security module to secure the Spring MVC application with the help of basic and form based security.

Chapter 8, *Versioning and Deploying*, shows us how to collaborate on the application, which has been developed in parts, or by different team members simultaneously. In this chapter, we will set up and integrate Tortoise SVN as a versioning tool used to collaborate on the code. We will also discuss the creation of repositories, users, and setting access rules for Collabnet and Visual SVN servers.

What you need for this book

You will need to have sound knowledge and practical exposure of core Java to understand this book. Along with this, knowledge of basic JDBC and the concepts of object-oriented programming language is required. As we are using Eclipse IDE throughout the book, you should be familiar with it. Those who have an introductory knowledge of Spring beginner framework can refer to this book easily. If you are beginner for Spring, we suggest you first go through the basic concepts of Spring configuration and get some practical experience. A basic knowledge of Hibernate and JUnit will be an added advantage.

Who this book is for

The idea for this book is to give the reader experience of creating an application step by step using Java modular programming step by step. The book is useful for any novice developer who wants to get exposure of modular Java development. The book is also useful for anyone who wants to have a roadmap for developing an application in stages such as problem statement, UI development, business logic development, database layer development, and so on. The book covers all the aspects of application development required into a turn a the problem statement to product, with coverage of security, maintaining versions, and the deployment process.

Conventions

In this book, you will find a number of text styles that distinguish between different kinds of information. Here are some examples of these styles and an explanation of their meaning.

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: "Every object created based on this type will inherit these default properties such as `toString`, `valueOf`, `hasOwnProperty`, and so on."

A block of code is set as follows:

```
function doAddition(num1, num2){
  return num1 + num2;
}
function doSubtraction(num1, num2){
  var result = null;
  if(num1 > num2){
  result = num1 - num2;

  }else{
    result = num2 - num1;
  }
  return result;
}
```

[Warnings or important notes appear in a box like this.]

[Tips and tricks appear like this.]

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or disliked. Reader feedback is important for us as it helps us develop titles that you will really get the most out of.

To send us general feedback, simply e-mail `feedback@packtpub.com`, and mention the book's title in the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide at `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for this book from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

You can download the code files by following these steps:

1. Log in or register to our website using your e-mail address and password.
2. Hover the mouse pointer on the **SUPPORT** tab at the top.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box.
5. Select the book for which you're looking to download the code files.
6. Choose from the drop-down menu where you purchased this book from.
7. Click on **Code Download**.

You can also download the code files by clicking on the **Code Files** button on the book's webpage at the Packt Publishing website. This page can be accessed by entering the book's name in the **Search** box. Please note that you need to be logged in to your Packt account.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR / 7-Zip for Windows
- Zipeg / iZip / UnRarX for Mac
- 7-Zip / PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Learning-Modular-Java-Programming`. We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you could report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **Errata Submission Form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded to our website or added to any list of existing errata under the Errata section of that title.

To view the previously submitted errata, go to `https://www.packtpub.com/books/content/support` and enter the name of the book in the search field. The required information will appear under the **Errata** section.

Piracy

Piracy of copyrighted material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works in any form on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors and our ability to bring you valuable content.

Questions

If you have a problem with any aspect of this book, you can contact us at `questions@packtpub.com`, and we will do our best to address the problem.

1
Introducing Modular Programming

Software development is a complex, time-consuming process, where success depends on teamwork. We keep on talking about software or software development. Sometimes we are part of the process as well. But we will be in one of the roles as architect, developer, tester, or deployer. Though we are always concentrating on a role, knowing the overall process always benefits us.

In this chapter, we will be going through the following topics:

- What is software and software development?
- What are enterprise applications?
- The role of modular programming in enterprise applications
- Introduction to and the importance of versioning

Software – the perspective

A software application is a program which enables end users to perform a specific task, for example, online money transfer, and withdrawal of money from an ATM or the use of an Eclipse to develop an application. This application is complex, scalable, and distributed, providing a complete solution to the end user. Applications known as enterprise applications are needs-based, providing solutions to business requirements rather than to an individual. The organization will use this application or integrate it within an existing application.

Enterprise applications may vary from business to business, for example, school or employee management systems, banking applications, online shopping applications, or e-commerce applications. All such enterprise applications provide displaying, processing, and storing data as their basic feature. Along with these features, the application can also provide transaction management and security services as advanced features. We access such applications typically through a network system rather than on an individual machine.

Let's briefly discuss the software development process before moving ahead:

- The software is always a solution or part of the solution to an enterprise problem. A good start in the development process is knowing exactly what the expectations are from the software, what types of solutions need to be included, what the data input will be, and what the output from the application is. This phase will be called the **requirement collection phase**.
- Once we get an idea about the requirements, now it's time to decide the hardware specification, the system requirements, the architecture to use, the design to follow, and so on. This phase is called **designing**.
- Using the design document, now developers will come in action to start a very important phase called **development**, where the actual coding takes place.
- Suppose we have developed a product; how do we prove that it is the right solution for the requirements which we got in the first phase? Yes, with the help of **testing**. We can carry out unit testing, integration testing, assembly testing, and acceptance testing to ensure that the requirement has been met.
- After successful testing, now it's time for the user to use it. This is nothing but the **deployment** phase, after which it is ready for use.
- Talking in terms of one phase after deployment, the work is over but what if any runtime issue emerges? What if the client recommends some minor or major changes? Or what if it has a bug? Because of this, post-deployment is also a very important step, which we call **maintenance**.

Although these phases theoretically come one after another, there can be different approaches called **software development process models**, such as the waterfall model, iterative model, spiral model, v model, agile model, and so on.

Modules

Application development is composed of many interconnected parts which interact with each other. To withstand high market demand and increasing competition, software should have a good look and feel, and ease of use. To develop a compatible solution, the developer has to think about compound structure as well as user perspective. It's quite difficult to develop such a product single-handed. It's teamwork, where the development is running alongside. The team members will build up separate small modules dedicated to part of the actual solution. These small modules will be clubbed together and interact with each other to form a complete solution.

What is behind and in a module?

Each module which has been developed will be performing a unique responsibility. When a module is responsible for a single task, it will be called **cohesive**. The cohesiveness will make the module more maintainable. Also, it will be less frequently changed. A good design perspective is to try writing a module which will be highly cohesive.

The two modules developed separately will now need to have interaction. To make them interactive, we have to introduce them. This will be done by making them dependent on each other. This dependency is termed **coupling**. When the code size and number of modules are small, coupling won't be a problem. But in an enterprise application, the code size is huge. Any little change makes a difference and then all of its dependencies should be changed accordingly at a number of places. This makes the code unmanageable. So it's always recommended to have loosely coupled modules.

The practical aspect

Let's take the example of a desktop, the one which we use in our routine. A desktop consists of a monitor, CPU, keyboard, and mouse. If a new monitor with some advanced features is introduced in the market, what we will do? Will we buy a new desktop or just replace the monitor?

As per the convenience and also the cost, it's feasible to just replace the monitor and not the whole desktop; how come this is possible? It's possible because the desktop is assembled with subunits, which are easily replaceable. Each subunit is cohesive for the work and they are not tightly coupled. This happens when we use modularization. When we write an application that uses similar concepts, it is called **modular programming**.

The gang – modular programming

Modular programming is the process of dividing a problem into smaller subunits and then making them interact with each other. Each subunit will revolve around a part of the problem. These subparts are quite easily reusable or replaceable. This designing technique gives a helping hand to the developers to develop their individual units and later combine them. Each subpart can be termed a module. The developers do not need to know what the other modules are or how they have been developed. Modularizing the problem will help the developers to achieve high cohesion.

The world of modules

The pluggable component which can be easily integrated into the application will provide the solution to a particular problem. For example, we want an Eclipse to support **Subversion** (**SVN**) (one of the versioning tools). Now, we have two choices. One, to start the development of Eclipse again from scratch or, two, to develop an SVN application. If we choose the first choice, it's very time-consuming and we already have Eclipse in a working condition. Then why start it from scratch? Yes, it's a very bad idea. But it would be great to have an SVN solution to be developed separately which is an SVN plugin; this plug-in can be easily integrated into eclipse. See how easily the two modules— eclipse, which was in working and the new SVN module—have been integrated. As SVN is a separate module, it can be integrated with NetBeans (one of the IDEs). If we had developed it in eclipse, then it would not be possible to integrate it in any other IDE. When we develop any application, from our point of view, it's always the best. But being a good developer, we need to be sure of it. How to check whether the application we have developed is working fine for the aspects or not? Yes, we need to test it, whether each part is working correctly or not. But is it really so simple? No, it's not. Not just because of complicated logic but due to its dependency. Dependency is a factor which is not under the control of the developer. For example, we need to check whether my credentials are correct or not when I am trying to login. We don't want to test the tables where the data is stored, but we want to check whether the logic of tracking the data is correct or not. As we have developed a separate data access module, the testing becomes easy. In Java, a single functionality can be tested with JUnit.

Testing helps the developer to test an application which processes the data and provides an output. If we are getting the correct result, we do not have a problem, but what if the output is wrong? The process of finding bugs in a module is called debugging. The debugging helps us to find defects in an application. That means we need to track the flow and then find out where the problem started. It's quite difficult to find the defect if the system is without modules or if the modules are tightly coupled. So it's good programming practice to write an application which consists of highly cohesive, loosely coupled modules.

There is one more fact which we need to discuss here. Sometimes, when the actual main development is progressing, we come across a point where we actually want to add a new feature. This new feature was not added while the basic discussion was going on; here, we want to do parallel development. In this case, we don't want to replace the previous development but we want to support or enhance it. As our application consists of modules, a developer can go ahead as most of these modules are independent and can be reused.

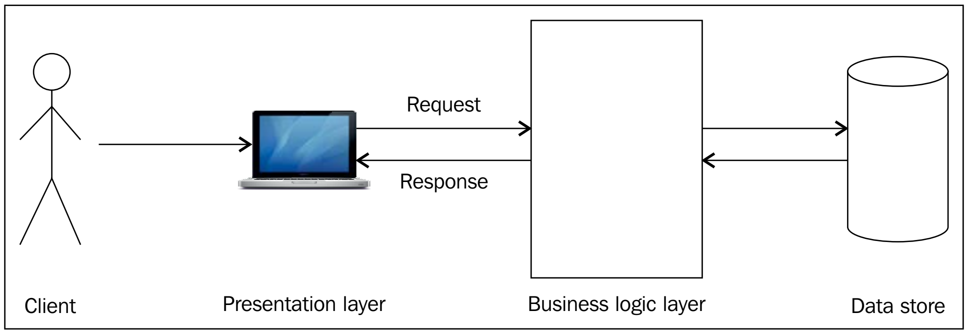
Tiers and layers in an enterprise application

An enterprise application is an application which has been developed to fulfill the requirements of a business. Being an enterprise application, it normally has huge code. Maintaining such huge code all together is a very complex task. Also, developing the code takes lots of time. So the code is been divided into small, maintainable modules which can be easily developed separately and later on combined to give a final product. All modules which provide similar kind of functionality will be grouped together to form a **layer**. These layers are the logical separation of modules. But sometimes, for better performance, one layer can be also spread over the network.

Layers are a logical separation of the code to increase maintainability. But when we physically move one typical layer and deploy it on another machine, then it will be called as a **tier**. At any one time, many users will be using the enterprise application simultaneously, so the use of a tiered application provides good performance.

Let's consider a web module for login. The user will open the browser and the login page will be rendered. The user will enter their credentials (username and password). After submitting the form, a request will be sent to the server to perform the authentication. Once the data is received on the server side, the business logic layer will process the data and put the result in the response. The result depends on whether the credentials are present in database or not.

Finally, the response will be generated and the result will be sent back to the browser. Here, the user interface, business logic, and database are the three distinct features involved. These are called as the presentation layer, business logic layer, and data storage layer, respectively.



Layers in an enterprise application

Each of these layers will talk with the above layer and exchange data. The process broadly takes place as follows:

- The user will open the browser and hit the URL.
- A login page will be rendered on the browser. The user will fill in the form and submit it to be processed by the business logic layer.
- To check the data, the business logic layer will communicate with the data storage layer.
- According to the result returned from the data storage layer, the business logic layer will now send the result to the presentation layer and the client's browser will render the results page.

Now we understand the difference between a tier and a layer, let's discuss tiers in detail. Depending on how many physical separations one application is using, it will be called a one-tier, two-tier, or multi-tier application.