SCJD Exam with J2SE 5

Second Edition

Andrew Monkhouse and Terry Camerlengo

Apress®

SCJD Exam with J2SE 5, Second Edition

ISBN (pbk): 1-59059-516-5

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jason Gilmore
Technical Reviewer: Jim Yingst
Editorial Board: Steve Anglin, Dan Appleman, Ewan Buckingham, Gary Cornell, Tony Davis, Jason Gilmore, Jonathan Hassell, Chris Mills, Dominic Shakeshaft, Jim Sumser
Project Manager: Beth Christmas
Copy Edit Manager: Nicole LeClerc
Copy Editor: Liz Welch
Assistant Production Director: Kari Brooks-Copony
Production Editor: Lori Bring
Compositor: Dina Quan
Proofreader: Elizabeth Berry
Indexer: John Collin
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 3

Project Overview

In this chapter, we introduce the sample application, which will serve as a wellspring for the myriad of topics required for developer certification. The sample project, Denny's DVDs, has a structure and format similar to the one you will encounter during the Sun Certified Java Developer (SCJD) exam, and it will demonstrate each of the essential concepts necessary for successful completion of the certification project. Each chapter adds an integral component to the project and builds from the preceding chapter, so that by the end of the book you will have a complete and properly functioning version of Denny's DVDs version 2.0. As an added benefit, Denny's DVDs utilizes Java 2 Platform Standard Edition (J2SE) 5, and some of the "Tiger" features, such as autoboxing and generics, are elucidated in the following chapters.

Note "Tiger" is the code name for J2SE 5, and the two terms will be referred to interchangeably throughout the book. Sometimes we will use the term "Tigerize," which means to add a J2SE 5 language feature to code originally composed as a J2SE 1.4 program. We do this quite a bit in the sample project, and even the `DOSClient` makes use of generics. For more information on J2SE 5 and the plethora of new features that have been added (some of which are discussed in this book), go to `http://java.sun.com/developer/technicalArticles/releases/j2se15`.

What Are the Essential Requirements for the Sun Certification Project?

To demonstrate "developer-level" competency for Sun certification, your project submission must successfully accomplish the following objectives:

- It must implement an application interface provided by Sun.
- It must use either RMI or serialized objects over sockets for networking.
- It must use Swing with a `JTable` for display.
- It must be entirely contained within one single executable JAR and, consequently, should not require any command-line options to run.
- Configuration settings must be persisted between application runs.

Your project submission must consist of these components:

- An executable JAR file, which will run both the stand-alone client and the network-connected client
- An executable server-specific JAR file, which will run the networked server
- A common JAR file, which will contain code common to both client and server applications:
  - An `src` directory containing the source files for the project
  - The original data file supplied with the instructions
- A `docs` directory that will contain
  - The API generated by Javadoc
  - The end-user documentation
- The file summarizing your design choices

The entire submission should be packaged in a JAR file, as described in Chapter 9.

■ **Note** JARs were initially created to allow applets to be downloaded in a single HTTP request, rather than multiple round-trips (request-response pairings) to retrieve each applet component, such as class files, images, and so forth. Executable JAR files have file associations so that clicking on them will run `javaw -jar` on Windows and `java -jar` on Unix.

The Denny's DVDs sample will eventually be bundled into one JAR file. That JAR file will contain an executable JAR file named `runme.jar`, containing the database files, documentation, and source code. The `runme.jar` will handle both the server and the client depending on how it is invoked. Running the final application from the executable JAR file will be explained in Chapter 9.

■ **Note** The previous edition of this book included a chapter on Java's New I/O, or NIO. At the time, NIO was a new 1.4 topic, and the extent that NIO could be used for the certification exam was unclear. Subsequently, after the publication of the first edition and the release of J2SE 1.4, Sun explicitly disallowed the use of NIO as a networking solution for the certification project, but permitted its use as a mechanism for file I/O. Admittedly, the main reason for including NIO in the previous edition was just to demonstrate this cool new technology. Our primary examples (downloadable from the Apress web site) did not use NIO in the networking layer. Unfortunately, there was some confusion since many believed that the inclusion of NIO indicated that our proposed solution required NIO. It did not. For this reason, we have decided to drop the discussion of NIO in the current edition of this book to avoid any further confusion, but would like to make it clear that sockets or RMI are required for the networking layer but that channels can safely be used for plain old file I/O.

Introducing the Sample Project

Denny's DVDs is a DVD rental store for a small community. The certification project requires that the application be built on a provided interface. Sun will include an interface as part of the assignment, and you will be responsible for implementing that interface and developing a fully featured application. Our sample project will take the same approach. We will imagine that the infamous Denny of Denny's DVDs will define an interface that he expects the developer (i.e., you the reader) to implement. That interface, `DBClient.java`, will be our starting point for development. Figure 3.1 shows a UML use case diagram for the primary operations the system must support.

■ **Note** The acronym *UPC* will be bandied around quite a bit, so a little background information may come in handy. UPC, which stands for Universal Product Code, is an official designation for a product, whether it be a book, a CD, or a DVD. It is a unique number identifying that product worldwide. For more trivia-related information on UPC (and related concepts such as European Article Numbering [EAN] and checksums), refer to such sites as Wikipedia (`http://en.wikipedia.org/wiki/Main_Page`) and the various online UPC databases. Of course, such a topic is way outside the scope of this book. For our purposes, UPC is a nice surrogate for a system-wide identifier or primary key.

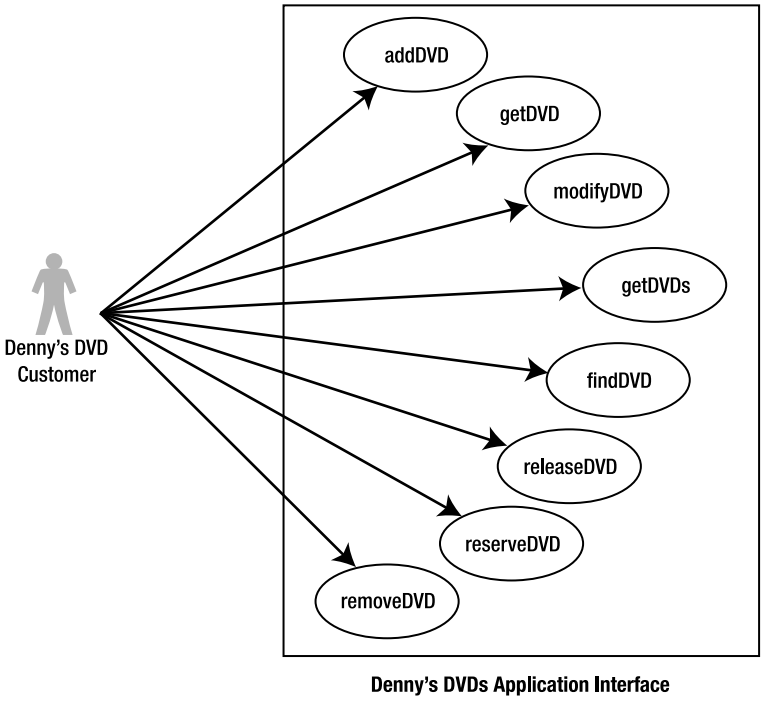

Figure 3-1. *Denny's DVDs use case diagram*

The operations in Figure 3.1 make up the public interface, `DBClient.java`, that the Denny's DVDs application must implement. Here are descriptions of the interface methods:

`addDVD`—Requires a DVD object as an input parameter. Will add the DVD to the system if the UPC is unique; otherwise, will throw an exception.

`getDVD`—Given a UPC value for a DVD, this method should return the corresponding DVD object. A null object is returned if the UPC is not found in the database.

`modifyDVD`—Requires a DVD object with a UPC that exists in the database. Will overwrite an existing DVD with new DVD values. If the database does not contain a DVD with the supplied UPC code, then false is returned and the database is not updated.

`getDVDs`—Should return a collection of all the DVDs in the database. If there are no DVDs in the database, then an empty collection should be returned.

■ **Note** The Denny's DVDs system available for download comes with a starter database, which is referred to throughout the book's examples.

`findDVD`—This operation should use a regular expression query as the input parameter. The supplied input parameter should match on multiple attributes with a regular expression query. The input parameter is the regular expression query, so the application must translate the user's search criteria into a regular expression query prior to invoking this method. Transforming the user's search criteria into a regular expression can be done either in the GUI or via some intermediate class, but not in the `findDVD` method itself. Of course, you must not require or expect the user to enter the regular expression syntax as the search criteria. A collection of DVDs should be returned.

`releaseDVD`—Will increment the count of available DVDs. This operation is equivalent to a rental return. The operation should return true, which indicates whether the DVD identified by the UPC exists in the database and has copies out for rental. Otherwise, the operation returns false.

`reserveDVD`—Will decrement the DVD count indicating that someone has rented one of the available copies. Requires a UPC as an input parameter. If the UPC of the DVD to reserve does not exist, false is returned; false is also returned if no more copies of the DVD are available for rental.

`removeDVD`—Will remove the DVD with the matching UPC and return true. If the UPC is not found in the database, then no DVD is removed and the method returns false.

The interface must have a GUI to allow the execution of each of the required methods listed here. Also, the GUI must be capable of connecting to a server on a network, or work in stand-alone mode and connect to a server on the localhost. You must also consider design issues related to concurrent user access and record locking. Listing 3.1 contains the code for the `DBClient.java` interface.

■ **Tip** Before inspecting the Denny's DVDs `DBClient.java` source code available for download, try writing the class yourself and see how close your interface is to the one used in the sample project. Since you will also be given the interface in the actual assignment, this exercise should be performed just for fun.

Listing 3-1. *The DBClient Interface*

```
package sampleproject.db;

import java.io.*;
import java.util.regex.*;
import java.util.*;

/**
* An interface implemented by classes that provide access to the DVD
* data store, including DVDDatabase.
*
* @author Denny's DVDs
* @version 2.0
 */

public interface DBClient {

    /**
     * Adds a DVD to the database or inventory.

     *
     * @param dvd The DVD item to add to inventory.
     * @return Indicates the success/failure of the add operation.
     * @throws IOException Indicates there is a problem accessing the database.
     */
    public boolean addDVD(DVD dvd) throws IOException;

    /**
     * Locates a DVD using the UPC identification number.
     *
     * @param UPC The UPC of the DVD to locate.
     * @return The DVD object which matches the UPC.
     * @throws IOException if there is a problem accessing the data.
     */
    public DVD getDVD(String UPC)throws IOException;

    /**
     * Changes existing information of a DVD item.
     * Modifications can occur on any of the attributes of DVD except UPC.
     * The UPC is used to identify the DVD to be modified.
```

```
     *
     * @param dvd The DVD to modify.
     * @return Returns true if the DVD was found and modified.
     * @throws IOException Indicates there is a problem accessing the data.
     */
    public  boolean modifyDVD(DVD dvd) throws IOException;

    /**
     * Removes DVDs from inventory using the unique UPC.
     *
     * @param UPC The UPC or key of the DVD to be removed.
     * @return Returns true if the UPC was found and the DVD was removed.
     * @throws IOException Indicates there is a problem accessing the data.
     */
    public  boolean removeDVD(String UPC) throws IOException;

    /**
     * Gets the store's inventory.
     * All of the DVDs in the system.
     *
     * @return A List containing all found DVD's.
     * @throws IOException Indicates there is a problem accessing the data.
     */
    public List<DVD> getDVDs() throws IOException;

     /**
      * A properly formatted <code>String</code> expressions returns all
      * matching DVD items. The <code>String</code> must be formatted as a
      * regular expression.
      *
      * @param query The formatted regular expression used as the search
      * criteria.
      * @return The list of DVDs that match the query. Can be an empty
      * Collection.
      * @throws IOException Indicates there is a problem accessing the data.
      * @throws PatternSyntaxException Indicates there is a syntax problem in
      * the regular expression.
     */
    public Collection<DVD> findDVD(String query)
            throws IOException, PatternSyntaxException;

    /**
     * Lock the requested DVD. This method blocks until the lock succeeds,
     * or for a maximum of 5 seconds, whichever comes first.
     *
     * @param UPC The UPC of the DVD to reserve
```

```
     * @throws InterruptedException Indicates the thread is interrupted.
     * @throws IOException on any network problem
     */
    boolean reserveDVD(String UPC) throws IOException, InterruptedException;

    /**
     * Unlock the requested record. Ignored if the caller does not have
     * a current lock on the requested record.
     *
     * @param UPC The UPC of the DVD to release
     * @throws IOException on any network problem
     */
    void releaseDVD(String UPC) throws IOException;

}
```

■ **Note** The sample project is not a full-featured e-commerce system. Instead, think of it as a program that will demonstrate the concepts needed to successfully complete the project portion of the SCJD exam.

Application Overview

So, what's next? In this section, we present an overview of the new system and describe the steps necessary to successfully implement the sample project.

Architecturally, the application is a traditional client-server system composed of three key parts: the server-side database with network server functionality, the client-side GUI, and a client-side database interface that handles the networking on behalf of the user interface. Figure 3-2 shows a high-level overview of the new system.

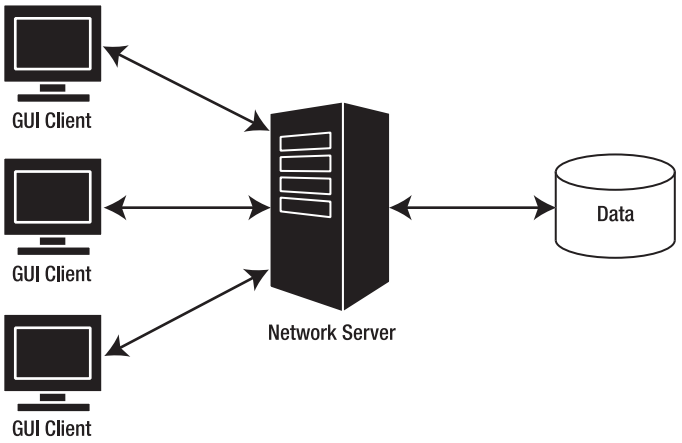

Figure 3-2. *Denny's DVDs system overview*

The users, Denny's employees, must be able to do the following:

- Perform the operations defined in the `DbClient.java` through a GUI interface.
- Enable multiuser networked access to a centralized DVD database.
- Allow network access via RMI or sockets.
- Make the database implementation thread-safe (implicitly required for multiuser networked access).

■ **Tip** Included in the source code available for download is a `DOSClient.java` class. This is a useful command-line program that was available in the download that accompanied the first edition of this book but, due to recent changes in the format of the certification project, is not necessary in the second-edition version of Denny's DVDs. Even though we do not discuss the `DOSClient.java` in this book, we include it anyway as a convenience to those downloading the code. Think of the `DOSClient.java` as a simplified command-line version of the GUI tool.

Creating the GUI

The GUI must allow an employee to view DVD information such as the UPC, title, rental status, director, actors, actresses, composer, and number of store copies. The GUI must also allow an employee to conduct a search on any of these DVD attributes. These attributes are listed in the class diagrams in Figure 3-3. The GUI should provide the user with the option of connecting to the database locally or through a network. It does not need to take into account any security features such as authentication and logon. Because the SCJD exam currently requires the use of Swing for the GUI, you will also use Swing for your project's GUI.

■ **Note** Swing is a Sun technology built on top of the Abstract Windowing Toolkit (AWT). AWT is part of the Java Foundation Classes (JFC). Knowledge of the AWT is no longer necessary for programmer-level certification, and a direct understanding of how Swing utilizes the AWT is not essential for developer certification.

DVD
upc name composer director copy leadActor supportingActor year
getUPC() setUPC(String upc) getName() setName(String name) getComposer() setComposer(String composer) getDirector() setDirector(String director) getCopy() setCopy(int copy) getLeadActor() setLeadActor(String leadActor) get SupportingActor() setSupportingActor(String supportingActor) getYear() setYear(String year) setRented(boolean renting)

Figure 3-3. *Class diagrams*

Network Server Functionality for the Database System

Version 1.0 of Denny's rental-tracking system can be run either locally or across a network. In local mode, the GUI will connect to the database only if it is located on the same machine. In network mode, the GUI should connect to the data server from any machine accessible on the network. The network implementation can make use of either sockets or RMI (see Figure 3-4). We demonstrate both approaches in Chapters 5 and 6.

■ **Note** Even though there isn't a version 2.0 discussed in this book, we will still refer to the system we are describing as Denny's DVDs version 1.0. Perhaps version 2.0 will show up in a third edition.

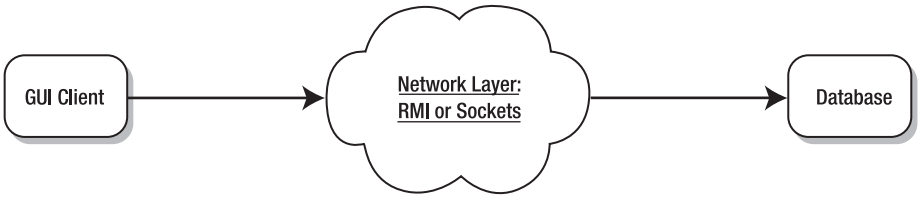

Figure 3-4. *High-level client server functionality*

Because it is possible for multiple clients to connect over a network and attempt to modify the same records simultaneously, the application must be thread-safe. We discuss thread safety in more detail in Chapter 4. For now, it will suffice to know that thread-safe code protects an object's state in situations where multiple clients are accessing and modifying the same object. It is your responsibility to make sure that your certification submission is thread-safe, as you'll learn in Chapter 4 in the section, "Understanding Thread Safety."

There is no need to notify clients of *nonrepeatable reads*—that is, it is not a requirement that all clients viewing a record that has been modified be notified of the modification. However, if two employees attempt to rent the same DVD simultaneously, then only one customer will get the DVD. This is referred to as *record locking*. In Chapter 4, record locking will be explored in more detail (with an example given in Chapter 5).

The application should be able to work in a non-networked mode. In this mode, the database and user interface run in the same virtual machine (VM), no networking is performed, and no sockets should be created. Later, in Chapter 5 (which covers networking), separate implementations for both RMI and sockets are demonstrated.

Summary

In this chapter, we introduced the public interface of Denny's DVDs rental-tracking program. We discussed the project requirements and each method of the interface that you will be responsible for implementing. All of the code for the sample application can be obtained from the Source Code section of the Apress web site (`http://www.apress.com`).

The new system should be accessible by multiple clients either across a network or locally. The GUI should be intuitive and easy-to-use, and the application must be thread-safe. The remainder of this book examines the requirements covered in this chapter. Denny's DVDs rental-tracking program will evolve as each new concept is introduced.

FAQs

Q The Sun assignment instructions tell us that we must include the instructions and data file in our submission. Why is this needed?

A There are multiple assignments available from Sun, and multiple versions of these assignments. Some of the changes in the assignment versions include variations in the data file format, the provided interface, the required class names, or any combination of these variations. Providing both the instructions *you* implemented and *your* data file ensures that the assessor will be comparing your submission with your instructions, and not anyone else's. Likewise, this means that the assessor is guaranteed to receive the correct instructions at the same time he or she receives your submission.

Q The provided interface does not include an exception I would like to throw. Can I add it?

A No. As stated in the instructions, other applications are expecting to use this interface, so if you add exceptions, the other application will need to catch them. This could prevent the other application from working.

Q I think this application will work better if I add another method to the interface. Can I do this?

A You can; however, we advise against it for the Sun assignment. Adding new interface methods will break the contract between the data formats and user interface. Additionally, if you do this on the actual assignment you may end up failing. A better approach is to add methods to the implemented classes. We encourage you to add class methods (that is, not interface methods provided by Sun) or to modify implementations as a way to better understand the project and source code.

Q How similar is this chapter's example to the one Sun will provide?

A There is no guarantee what future exams will look like from Sun, but this sample demonstrates the concepts required to achieve the Java developer certification, and it also introduces new features in J2SE.

Q How should I use the provided code samples?

A The completed version of the Denny's DVDs system is available for download. Each chapter will describe the evolution of the project from the required `DBClient` interface developed in this chapter to the remainder of the application. As you read the book, refer to the relevant section of the code base in order to understand the full implementations. Chapter 9 explains in detail how to compile and execute the entire Denny's DVDs system in a manner similar to the way your actual Sun submission will need to be executed.

An alternative approach is to study the sample project source code first and then read the book to provide an explanation for the design choices made and how those choices relate to similar decisions you will encounter while developing the actual Sun certification project. We recommend reading the book first and then referring to the code, but the alternative approach of reading the code and referring to the book should also produce successful results. It really depends on which approach makes more sense for you and your natural learning methods.

Q Can I add features not discussed in this book?

A Of course you are free to modify the project and make it more sophisticated. In fact, inquisitiveness and a sense of experimentation are very important qualities for a software engineer. However, the project has been carefully designed with two goals in mind. The first and more important goal is to cover all of the concepts required for the SCJD exam. So if you do decide to ad-lib, keep in mind the purpose of the sample project. The second goal is to introduce the new features of J2SE 5, such as autoboxing and generics.