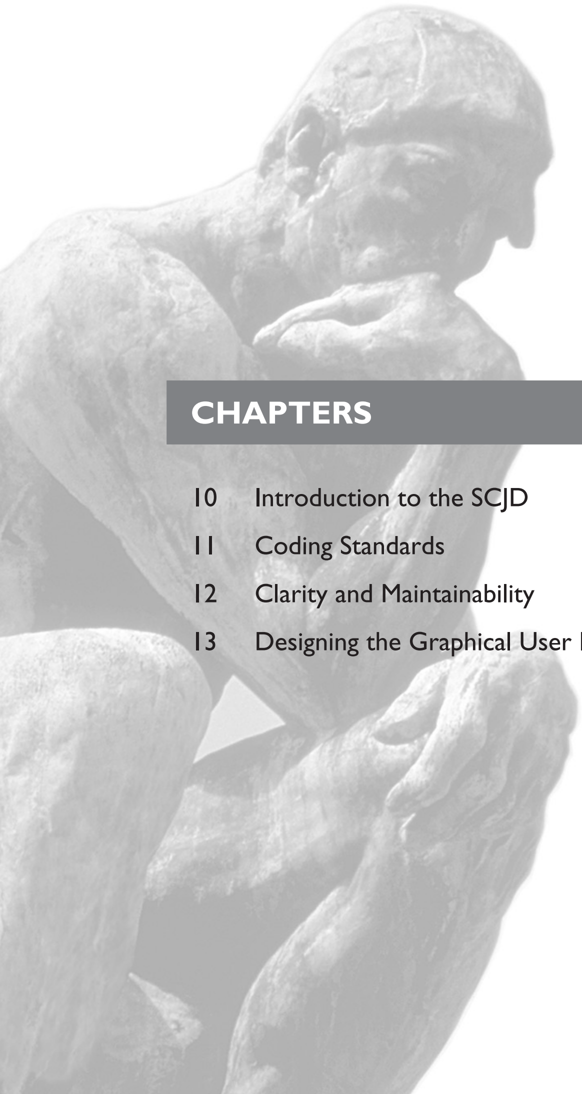
JAVA™ 2

SUN™ CERTIFIED PROGRAMMER & DEVELOPER

Part II

The Developer's Exam



CHAPTERS

JAVA™ 2

SUN™ CERTIFIED PROGRAMMER & DEVELOPER

10 Introduction to the SCJD

CERTIFICATION OBJECTIVES

- Understand the Sun Certified Java Developer Exam Process

CERTIFICATION OBJECTIVE

Understand the Sun Certified Java Developer Exam Process

OK, so now you know everything about the language. But can you actually *build* something in it? You'll hear that argument from some who've never taken (or passed) the programmer's exam. Obviously, they don't understand how darn difficult the programmer's exam actually is, but nonetheless there *is* something to the claim that, "just because you know how the compiler and VM work does not mean you can develope software." The Developer exam, which is unique in the IT exam world, lets you answer that question (most often posed by a prospective employer).

In the Developer exam, you get to put your code where your mouth is by developing a software application. In fact, the Developer exam isn't even a multiple-choice test but rather a project that you build, given a (somewhat sketchy) specification. You're told what to build, with some general guidelines, and then it's up to you to implement and deliver the program. You have an unlimited amount of time in which to finish the project (as of this writing), but there *is* a short follow-up essay exam (taken at an authorized testing center, just as the Programmer exam is). The follow-up questions are largely used to verify that it was *you* (not your hotshot programmer brother-in-law who owed you big time) who did the work. In other words, the follow-up exam asks essay questions that only the project developer could answer (for example, "Justify your design choice on…").

First, we'll lay out the facts of the exam—how it works, how you do it, etc., and then we'll dive into what you need to know to pass it. Keep in mind that the actual knowledge you need to pass cannot be stuffed into a book this size, even if we made the book big enough to crush a car. Being a programmer is one thing, but being a *developer* is quite another. And you can't become a developer just by memorizing some facts. Study and memorization can work for passing the Programmer's exam—but that's OK because the programmer's exam is designed to verify that you're smart and that you really know the language. A prospective employer doesn't have to train you in Java if you've passed the programmer's exam. But if your employer wants to verify that you can follow a spec and implement a well-designed, maintainable, *correct* application, then you need either previous experience successfully building one or more Java applications or you need to pass the SCJD.

The next seven chapters (in other words, the rest of the book) show you what you'll need to know and do to pass the exam, but it's up to you to do the heavy lifting. And unless you're already well-versed in some of the topics (Swing, Threads, RMI, etc.) then you'll need to do some outside reading and practice in those technologies. We're focusing here on what the exam assessors are looking for in your finished project.

How Does It Work?

The exam has two parts, The Assignment and The Essay. You must successfully pass both parts to become certified.

The Assignment

Once you register for the Developer's exam, you're given instructions for downloading your assignment. There are many possible assignments that you might get. The assignment is a 9- or 10-page document with instructions for completing the project. Instructions include both the application specification and requirements for implementation and delivery. It also includes notes about how the application will be marked (evaluated, graded, assessed).

The Essay

Once you've submitted your assignment, you should immediately register for the essay portion of the certification. You can't register until after you've submitted your completed assignment, but the sooner the better once you have submitted it. You really want to take the essay portion while the application you just completed is still fresh in your mind. The essay portion will feel somewhat familiar to you—it takes place in an authorized Prometric testing center, just as the Programmer's exam does. You have 90 minutes to complete the essay portion, and it normally involves just a handful (about five) questions.

The Assessment

Once you've submitted both your assignment and the follow-up essay, the two pieces will be sent to the assessor for grading. It might be four weeks or so before you learn the results.

Are You a Good Candidate?

If you haven't yet passed the Programmer's (SCJP) exam, then stop right now and go get certified. You *must* pass the Programmer's exam before you're allowed to register for the Developer exam. So we figure that you'll read the first part of the book, take the Programmer's exam (passing, of course), then come back at some point and start reading this part. That means by the time you're reading this part, this book should already be dog-eared, marked-up, scratched, bent, and possibly dusty (from that dry spell between taking the Programmer's exam and going for the Developer exam).

If you got to this paragraph, then we assume you're already a Sun Certified Java Programmer. But are you ready for the Developer exam? Well, the good news is that you don't *need* to be ready when you *register* for the exam. You've got plenty of time to complete the assignment once you download it. So unlike the Programmer's exam, you don't have to wait until you're at top form for passing the exam. You can download the assignment, analyze what you'll need to learn to complete it, and then get to work. Sun (and most candidates) estimates that it takes between 90 and 120 hours of solid work to complete the exam, and that assumes you're already familiar with all the necessary technologies (networking, database, threads/locking, Swing, etc.). Some people work for three weeks straight, as if the project were their full-time job. Others work on it when they can, in their spare time, and might take several months to actually finish it. Of course, there's always the chance that you download it and discover you're way over your head and unlikely to get up-to-speed within a year. But if you've passed the Programmer's exam and you're willing to commit the time to work on it (plus whatever additional time you need to learn any required technologies you're not familiar with), then we say go for it…if you've got the money (we'll get to that in the next section).

Having said all that, we don't recommend registering until you've read the rest of this book. It'll give you a better idea of what's really involved, and you might decide to wait a while if you're still a beginner at some of these technologies. But if you're comfortable with at least three of the following, chances are you're ready to at least download the assignment:

- Swing and GUI design
- Networking issues: sockets and RMI
- Database issues: searching and record-locking
- Writing clear, maintainable code
- OO design and development

How Much Does It Cost?

All you need is $250 (US dollars) and you're in business…for the first part. The SCJD is in two parts, remember: the exam assignment (the specification that you download, implement, and submit) and the follow-up essay. The follow-up exam is an additional $150. So you're looking at $400 total to get your certification. There's no partial certification, so submitting your exam doesn't get you anywhere unless you successfully take the follow-up exam. In other words, you can't be certified without spending the $400.

How Long Does It Take?

As of this writing, there is no time limit specified for completing the assignment once you've downloaded it, but we don't advise waiting more than a year, as the requirements *could* change. Plus, there's a new requirement (although these requirements could change at any time so check the Sun website frequently at http://suned.sun.com for updates) that you must not use a version of Java that is deemed "out of date." The current definition of *out of date* is that your version must not have been superceded by a new production version for more than 18 months by the time you make your submission. What that means is that if your version has been out for less than 18 months, you're fine. If your version is older than 18 months (in other words, its official public release was more than 18 months ago), then the version released directly *after* your version must be less than 18 months old. So don't take forever is what we're saying, or you could find yourself rewriting your application. It's not good enough for your program to *run* on newer versions; you need to indicate in your exam which version you've compiled and tested on.

What's the Exam Deliverable?

Chapter 17 covers this in picky detail, but the short version is: a JAR file. As of this writing, you must submit the entire application, including compiled working classes, source code, and documentation in a single JAR file. Your assignment instructions will specify *exactly* how you must submit it and the most important rule is that *you must not deviate in any way from the submission instructions.*

Can I Develop with an IDE?

You can, but everything you submit must be *your own creation.* In other words, no auto-generated code. So use an IDE as an editor but not as a GUI-building tool or something that implements your networking. And whatever you do, be sure to test on a machine other than your development machine! When using an IDE (or not, but there's more of a danger when using an IDE) you can end up being sheltered and protected from things like classpath issues, which allow your program to run fine on your machine and then blow up (OK, just *fail* to run) at runtime on another system.

How Is It Graded?

Once you've completed both the assignment and the essay exam, an assessor takes both pieces and performs the assessment. Your project is first assumed to be correct and is given a starting point value (currently 155 points, but this could change). Then points are deducted through a variety of audits. For example, you might get 12 points deducted for issues with your coding conventions, and perhaps another 15 (out of a possible, say, 18 points) for problems with your record-locking or search algorithm. That subtracts 27 from your starting total of 155, and leaves you with 128 points. Currently, the exam requires 124 points to pass, so you'd be good with 128. Your instructions will give you an idea of the relative importance of certain evaluation (audit) criteria, but you won't know the specific values for specific violations. DISCLAIMER: the point values mentioned here are merely examples of how the exam is graded; they are *not* the actual point values used in the assessment. The only thing you will know with certainty is the relative importance of different aspects of your project. We'll give you one clue right now, though: code readability/clarity and threading/locking will be extremely important. You'll almost certainly find these two areas carrying the most weight on your assignment instructions.

What Are the Exam Assessors Thinking?

OK, we aren't mind readers, and for all we know the assessors are thinking about last night's Bellbottom Bowling party as they mark your exam. But we *do* know one thing: *they aren't looking to see how clever an algorithm designer you are!* If anything, it's just the opposite. When you think of the Developer exam, don't think Lone Ranger. Instead, think *Team Player*. And don't even *think* about showing off your programming prowess by revising the specification to do something even better and cooler than what's asked for. There's a saying we have in the software world, and it will serve you well to remember it while building your Developer project: "Code as if the next guy to maintain it is a homicidal maniac who knows where you live."

exam Watch

The exam assessors aren't thinking like my-algorithm-is-bigger-than-yours code mavericks. They aren't looking for the next great breakthrough in record-locking. They aren't even looking for new, creative implementations. They are looking for development that says, "I'm a thoughtful programmer. I care about maintainability. Readability is very important to me. I want everyone to understand what's going on with the least amount of effort. I write careful, correct code. My code might not be the snappiest, and it might even incur an extra few bytes in order to make something less complex, but my logic is simple, my code and my design are clear and implement the specification perfectly, I didn't reinvent the wheel anywhere, and gosh—wouldn't you just love to have me on your team?" If your project submission says all that about you, you're in great shape.

The exam assessor looks at your code first from an entirely selfish perspective by asking, "Is this easy for me to evaluate?" Chapters 16 and 17 offer insight into what you need to do to make the assessor's job easier. Trust us on this one—they'd rather be at the beach (or skiing, mountain-biking, taking a Martha Stewart crafts workshop) than spending unnecessary time figuring out how to get your assignment working. Beginning with the "refreshed" exam assignments at the end of 2002, the requirements changed to make the submission rules much more strict, in order to benefit the assessor. If at any time you neglect to follow even a single submission requirement—say, the directory structure of your project puts the user documentation in a different folder—you'll be failed on the spot. The assessor won't make *any* allowances for misplaced files, even if the program still runs perfectly. Don't make them go *looking* for something.

Another aspect of making the assessor's life easier is what you'll learn in Chapters 11 and 12. *The little things really matter!* For example, while you might think—if you indent your code four spaces—that an occasional three-space indentation here and there is OK, what's the harm in that? The harm is in readability, and while a couple of inconsistencies in indentations might not be a big deal, adhering to the Java Coding Conventions is absolutely crucial for others looking at your code… *especially* the assessor.

We've seen people fail the exam because they put the curly braces on the line *below* the method declaration rather than immediately following the declaration (on the same line), violating the official Java Coding Conventions. While this infraction *alone* probably might not cause you to fail, the points deducted for code convention violations might be the ones that sink you where you otherwise might have squeaked by. You don't get to make very many mistakes in this exam. Just because *your* manager or *your* co-workers are tolerant of a little sloppiness here and there, the assessor won't be. Reread the preceding Exam Watch, copy it down on a post-it note, and stick it onto your bathroom mirror. Each morning, say it to yourself, "I'm a thoughtful programmer. I care…" (except say the whole thing).

What Are the Exam Assessors NOT Thinking?

If your solution works—according to the spec—then even if the algorithms might be tweaked just a little more for efficiency, you probably won't be marked down—especially if the code is clear, maintainable, and gets the job done correctly. They're also not looking for one particular solution. There is no one *right* way to implement your assignment. There are a gazillion *wrong* ways, however, and we'll be looking at some of those throughout the rest of the book. But here's one that's guaranteed to kill you (both on the exam and in the real world): *deadlock*. Remember, we talked about threads in Chapter 8, and you'd better take it all very seriously. If there's even a chance that your design could lead to deadlock (it doesn't have to actually *cause* deadlock right before the assessor's eyes) then you can probably kiss that $400 goodbye.

The bottom line is that they're *not* looking for The Perfect Solution. But they're also not looking for innovative new approaches, regardless of how clever, when well-known patterns or other solutions exist. They're especially not looking for you to reinvent the wheel or write your own, say, new and improved set of classes to replace the perfectly working core library packages.

What's the Assignment Like?

We can't give you a *real* assignment from the actual exam, of course, but here are a couple of examples to give you the *flavor* of what the specification might look like. And don't be thinking these are outlandish examples; wait 'til you see the *real* ones.

WindRider Horse Cruises

WindRider Horse Cruises (WHC) offers a variety of unique vacation trips, all on horseback. Copying the cruise ship model, WHC has 4-day, 7-day, and 14-day cruises that include all the food, drinks, and partying you can handle. (Which, after four straight days on a horse won't be much.) WindRider has grown steadily from a two-person outfit offering one cruise a month to a busy operation with several cruises running simultaneously in different parts of the world. But while the business has grown, their cruise booking software hasn't kept pace. The WindRider CEO is acting as the company's IT director, but he has some quirks. He insists on keeping the entire application—*including the database server*—homegrown. In other words, he doesn't want to buy or use a database server written by anyone but his trusted friend Wilbur. Sadly, Wilbur sustained an injury while fulfilling his *other* WindRider duties (training horses to tolerate the disco music) and that's where you come in. Your job is to build the new WindRider booking software. One restriction is that you *must* use the WindRider's existing data file format. All cruise records must stay in that format because the accounting part of the company still has software that requires that format, and you're only updating the *booking* software.

Customers must be able to call in to one of the four booking offices and request a cruise. A customer service representative then uses the new booking application (the one you're going to write) to search for and then book an appropriate cruise to meet that customer's needs. Although the data file lives on one machine, back at the head office, the three other booking offices (and possibly more in the future) need to be able to access it over a standard TCP/IP network. So there's a danger that two customer service agents could be trying to book the same cruise slot at the same time. (A cruise slot is like a 'cabin' on a real seafaring cruise. So any given cruise might have anywhere between 6 to 12 slots, and each slot represents a record in the data file.) You'll have to make sure that this doesn't happen! Overbooking would be a Really Bad Thing. (Especially for the horse.)

So the people who interact with the actual software are the customer service agents. But it's the actual cruise customers who are making the requests. For example, a customer might phone up and say, "I'd like a 4- or 7-day Horse Cruise sometime in August 2003, in the United States." The customer service agent must then use the system to perform a search for that customer's needs. The application needs to provide a list of all possible matching cruises and then also allow the agent to reserve (*book*) a cruise slot for that customer.

You must use a Swing GUI, and WindRider's CEO just happens to be dating a Computer-Human Interaction specialist, so you can bet she'll be looking for all the right characteristics of a usable GUI.

For networking, you have the choice between RMI and using regular old Java TCP sockets (with serialized objects). It's really up to you to make that decision, but you'd better be prepared to explain why you chose what you chose.

The data file format is a little ugly, not comma-delimited or anything, just a bunch of fixed-length fields. And you *must* stick to this data file exactly. We'll send it to you so you can see exactly how it's formatted and start working with it in your development. You can't change a thing about what goes into a record. You can't add a field, can't reformat the data…nothing. Your job is simply to build the actual database server that accesses the data file to allow for searching, booking, adding new cruises, etc. Oh, and don't forget about those concurrent user issues—you must lock these records in some way during use.

From his hospital bed, Wilbur sketched out what the database interface should be, and you need to follow this exactly (although you can add more, but you must *at least* provide these two methods in your public interface to the database server).

```
public void updateRecord(String[] recordData, int whichRecord) throws
LockedRecordException, NoSuchRecordException;

public int[] findByCustomerCriteria(Criteria criteriaObject);
```

But then you still need to add the methods for deleting, locking, etc. And you'll have to create the custom Exceptions and decide what should go in the Criteria class (the thing you're going to use to search the database).

Your job, ultimately, is to deliver the following:

- The customer service GUI application that they use to search and book records in the database.
- The actual database server application—the thing that actually gets into the data file and takes care of locking, etc. This is most likely the piece the GUI interacts with.
- Networking functionality so that multiple users can access this, remotely.

Confused?

That's part of the idea. You need to think through the problems, think about *new* problems not addressed in this spec, and figure out how to solve them, even in the face of incomplete information. The real world isn't perfect. Specs never seem to be complete. And the person you need to ask for clarification never seems to be at his desk when you call. Oh, and there's nobody—and we do mean *nobody*—who will reassure you that you're on the right track by implementing a particular solution. You're just going to have to roll your sleeves up and answer your own "what about <insert some scenario> ?" questions.

And boy oh boy are there issues. Both raised and unraised by this specification. The majority of the rest of this book raises those issues and gives you a lot to think about. We can't give you solutions—there *aren't* any right solutions, remember—and it wouldn't be ethical to work out all the issues here. That's the whole *point* of the Developer exam! The actual coding is quite straightforward and fairly simple. It's not like you're writing the world's greatest neural network or artificial life program. But thinking about the true business issues—about what the customer might need, what the customer service agents need, and what the business itself needs, and then planning and implementing a solution—are what this certification is all about. You'll thank us one day. And don't forget, if you get frustrated, just remember how much you like us for getting you through the Programmer certification. Which we did, or of course you wouldn't be reading this far into the book!

Overview of the Developer Exam Chapters

We're going to cover a lot of ground here, some at a high level and some a little lower. The high-level areas are the places where you need to design solutions and discover potential problems. Locking issues, for example, are handled at a high level. We'll raise issues to get you thinking, but you'll have to come up with your own designs—after all, we have no way of *knowing* what your exact assignment will be. The lower-level areas are reserved for things you must do throughout your entire application—such as coding standards, OO design, documentation, etc., and for tools such as *javadoc* and Jar. We also cover GUI usability in some depth, but it will be up to you to work out the implementations. The following is a chapter-by-chapter look at what we cover in the rest of the book:

Chapter 11: Coding Standards

As we mentioned earlier, even the failure to indent properly or line up your comments can mean the difference between passing and failing the exam. We'll cover the relevant parts of the Java Coding Conventions that you must be *very* meticulous about in every single line in every single class in your application.

Chapter 12: Clarity and Maintainability

This is where the whole Team Work mentality (or, if you prefer, the homicidal maniac thing) comes in. We'll look at what makes your code easy to read and maintain (and conversely, what makes it a *pain* to read and maintain), and cover things like reducing logic complexity, appropriate error-handling, and adhering to some fundamental OO principles.

Chapter 13: GUI Usability

Don't you just hate it when you're working in an application that makes you type things like "yes" or "no" rather than providing radio buttons? Or what about a nonstandard menu—you know, without a File and Help menu? Or worse, no menu bar at all. Or one that bombards you with dialog boxes for every little move you make. Or things that *should* scroll that don't. Or when you resize a window and things land anywhere other than where they were before. A well-designed GUI must be usable, useful, and *not* clumsy. Fortunately, there are established human interface guidelines to help inform our design, and that's what we'll look at in this chapter.

Chapter 14: Networking Issues

Hmm, what to choose…RMI or sockets? We'll cover the main points and then—even though there is definitely *not* a right choice for the exam—we'll spend most of the time on our personal favorite, RMI. And why laziness isn't necessarily a bad trait in a programmer.

Chapter 15: Database Issues

We know, we know…in the real world surely someone would just *buy* a database. Heck, there are *free* ones out there. But for the purposes of assessing your development skills, thinking through (and implementing) the tricky and subtle issues of concurrency will do nicely. So pretend, for the time being, that there *is* no such thing as a database. Or that you're the first person to have to build one. We'll look at some of the things you'll need to be thinking about in your design, and how crucial threads are to your design and implementation.

Chapter 16: Exam Documentation

Remember when we said your job was to make the assessor's life easier? (The assessor representing both the end-user and the client and the project manager of this application.) Now's your chance to shine. Or not. We'll look at everything from status messages to comments, but most of the focus is on *javadoc*, which you *must* provide for *all* repeat *all* classes and interfaces.

Chapter 17: Final Submission and Essay

"Real Women Ship" the saying goes (or something like that) and now it's time for you to call your application *finished* and package it up in a nice JAR and send it out. We'll emphasize the importance of getting your directory structures just right, and what you'll need to do at the command-line to run from the JAR in a particular way.

Key Points Summary

No time like the present to get started. But before we jump into code conventions, here's a quick summary of the points from this chapter:

- The Developer exam is in two parts, the Assignment and the Essay.
- You must complete (and pass) both parts to become certified.
- The Assignment is a set of instructions for building and delivering the application.
- Once you've submitted your Assignment, you can register for the Essay portion of the exam.
- You're given a minimum of one year to complete the Assignment (from the time you register and download it).
- Most candidates take between 90 and 120 hours to complete the Assignment.
- You're given 90 minutes on the Essay portion of the exam.
- You must be a Sun Certified Java Programmer (for Java 2) in order to register for the Developer exam.
- The certification costs $400 total ($250 for the Assignment portion and $150 for the Essay).
- You can develop with an IDE, but you must not include any IDE-generated code in your project. Every line must be coded by you.
- The Assignment is graded by giving your application a starting number of points and then deducting points for violations including minor things (curly braces in the wrong place) and major things (locking doesn't work in every situation).

- The Essay is designed largely to verify that *you* are the one who completed the Assignment. You need to understand the key issues of your design and be prepared to justify your decisions.
- The exam Assessors are more interested in the clarity and maintainability of your code than they are in your clever algorithms.
- Think like a Team Player rather than a lone coding maverick, even if it means your design and implementation are sometimes slightly less efficient, but more easily understood by others.