**Certification**

## Java Developer Assignment 310-252A

01:42:05

Next Summary End Survey

**Item 1 of 1** Mark item for review

This document tells you what you need, and what you must do, to submit your solution to the Sun Certified Developer for the Java 2 Platform programming assignment. Read it carefully before you begin work on the solution. This document contains strict guidelines about the way the work is to be performed. These guidelines ensure consistency and fairness.
The application distribution is composed of:
• This document
• A URL to download a non-relational database file click here
Be sure to maintain a backup copy of all files related to your project, including the distribution files, until you receive your final grade in case something is lost or corrupted in the process.

**Important Note About Automatic Failure**
Where this document uses the word "must" an absolute requirement is being described. If you fail to adhere to such a requirement, your assignment will be failed automatically, and without further evaluation. It is therefore imperative that you pay close attention to any statement using the word "must" in this document. Portions of your submission will be analyzed by software; where a specific spelling or structure is required, even a slight deviation could result in automatic failure.

**Application Overview**
Background
URLyBird is a broker of discount hotel rooms. They sell accommodations for business and pleasure travellers at short notice, helping hotels to fill rooms that would otherwise be left empty. They take bookings only within 48 hours of the start of room occupancy. Currently, URLyBird sells the rooms over the phone using a team of customer service representatives (CSRs). The CSRs interact with an aging custom-written application that has been drawing increasing criticism from the CSRs. In the future, URLyBird wants to move into Internet-based marketing, and hopes to be able to accept bookings direct from customers over the web.
The company's IT director has decided to migrate the existing application to a Java technology based system. Initially, the system will support only the CSRs, although the hope is that this interim step will give them a starting point for migrating the system to the web. The IT director does not anticipate much reuse of the first Java technology system, but intends to use that system as a learning exercise before going on to a web based system.
The company's IT department has a data file that contains the essential information for the company, but because the data must continue to be manipulated for reports using another custom-written application, the new system must re-implement the database code from scratch without altering the data file format.
The new application, using the existing data file format, must allow the CSRs to generate a list of accommodations that match a customer's criteria. This is the project that you have been commissioned to implement.
What you must do
The following are the "top level" features that must be implemented:
• A client program with a graphical user interface that connects to the database
• A data access system that provides record locking and a flexible search mechanism
• Network server functionality for the database system
The work involves a number of design choices that have to be made. In all such cases, the following principles should be applied.

**Clarity and Maintainability**

A clear design, such as will be readily understood by junior programmers, will be preferred to a complex one, even if the complex one is a little more efficient. Code complexity, including nesting depth, argument passing, and the number of classes and interfaces, should be reasonable.

**Documentation**
The code itself should be as clear as possible; do not provide comments that do not add to the comprehensibility of the code. Awkward or complex code should have descriptive comments, and javadoc style comments must be used for each element of the public interface of each class. You must create a full suite of documentation for the classes of the completed project. This must be generated using the tool "javadoc" and must be in HTML format. Provide javadoc documentation for all classes you write.

You must provide basic user documentation. This should be sufficient to allow a user who is familiar with the broad purpose of the project to use the application. This documentation must be in one of these three formats:
• HTML
• Plain text (not a wordprocessor format)
• Within the application as a help system.

**Correctness**
Your project must conform to this specification. Features that deviate from specification will not receive full credit. You will not receive extra credit points for work beyond the requirements of the specification.
Use of Standard Elements
Use of functionality provided by the core Java classes will be preferred to your own implementation of that functionality, unless there is a specific advantage to providing your own implementation.

**Overall Architecture**
Major Components
The main architecture of the application must be a traditional client-server system. There are three key parts: the server-side data management system, the client-side GUI, and the network connection between the two.
Non-Networked Mode
The program must be able to work in a non-networked mode. In this mode, the database and GUI must run in the same VM and must perform no networking, must not use loopback networking, and must not involve the serialization of any objects when communicating between the GUI and database elements.
The operating mode is selected using the single command line argument that is permitted. Architecturally, this mode must use the database and GUI from the networked form, but must not use the network server code at all.
Network Communication Approach
You have a choice regarding the network connection protocol. You must use either serialized objects over a simple socket connection, or RMI. Both options are equally acceptable. Keep in mind that networking must be entirely bypassed in the non-networked mode.
Restrictions on RMI
To avoid unnecessary complexity in the marking environment certain restrictions are placed on solutions that use RMI. Specifically:

You must not require the use of an HTTP server.
You must not require the installation of a security manager.
You must provide all classes pre-installed so that no dynamic class downloading occurs.
You must use RMI over JRMP (do not use IIOP)

**The User Interface**
The user interface for this assignment must satisfy the following criteria:
• It must be composed exclusively with components from the Java Foundation Classes (Swing

components).
• It must allow the user to search the data for all records, or for records where the name and/or location fields exactly match values specified by the user.
• It must present search results in a JTable.
• It must allow the user to book a selected record, updating the database file accordingly.
Your user interface should be designed with the expectation of future functionality enhancements, and it should establish a framework that will support this with minimal disruption to the users when this occurs.

**Server**
Required Interface
Your data access class must be called "Data.java", must be in a package called "suncertify.db", and must implement the following interface:

```
package suncertify.db;


public interface DB
{
// Reads a record from the file. Returns an array where each
// element is a record value.
public String[] read(int recNo) throws RecordNotFoundException;


// Modifies the fields of a record. The new value for field n
// appears in data[n]. Throws SecurityException
// if the record is locked with a cookie other than lockCookie.
public void update(int recNo, String[] data, long lockCookie)
throws RecordNotFoundException, SecurityException;


// Deletes a record, making the record number and associated disk
// storage available for reuse.
// Throws SecurityException if the record is locked with a cookie
// other than lockCookie.
public void delete(int recNo, long lockCookie)
throws RecordNotFoundException, SecurityException;



// Returns an array of record numbers that match the specified
// criteria. Field n in the database file is described by
// criteria[n]. A null value in criteria[n] matches any field
// value. A non-null value in criteria[n] matches any field
// value that begins with criteria[n]. (For example, "Fred"
// matches "Fred" or "Freddy".)
public int[] find(String[] criteria);


// Creates a new record in the database (possibly reusing a
// deleted entry). Inserts the given data, and returns the record
// number of the new record.
public int create(String[] data) throws DuplicateKeyException;


// Locks a record so that it can only be updated or deleted by this client.
```

```
// Returned value is a cookie that must be used when the record is unlocked,
// updated, or deleted. If the specified record is already locked by a different
// client, the current thread gives up the CPU and consumes no CPU cycles until
// the record is unlocked.
public long lock(int recNo) throws RecordNotFoundException;


// Releases the lock on a record. Cookie must be the cookie
// returned when the record was locked; otherwise throws SecurityException.
public void unlock(int recNo, long cookie)
throws RecordNotFoundException, SecurityException;

}
```

Any unimplemented exceptions in this interface must all be created as member classes of the suncertify.db package. Each must have a zero argument constructor and a second constructor that takes a String that serves as the exception's description.
Any methods that throw RecordNotFoundException should do so if a specified record does not exist or is marked as deleted in the database file.

**Network Approaches**
Your choice of RMI or serialized objects will not affect your grade, but no other approach is acceptable. In either case, the program must allow the user to specify the location of the database, and it must also accept an indication that a local database is to be used, in which case, the networking must be bypassed entirely. No authentication is required for database access.

**Locking**
Your server must be capable of handling multiple concurrent requests, and as part of this capability, must provide locking functionality as specified in the interface provided above. You may assume that at any moment, at most one program is accessing the database file; therefore your locking system only needs to be concerned with multiple concurrent clients of your server. Any attempt to lock a resource that is already locked should cause the current thread to give up the CPU, consuming no CPU cycles until the desired resource becomes available.

**Data file Format**
The format of data in the database file is as follows:
Start of file 4 byte numeric, magic cookie value. Identifies this as a data file 4 byte numeric, total overall length in bytes of each record 2 byte numeric, number of fields in each record
Schema description section. Repeated for each field in a record: 2 byte numeric, length in bytes of field name n bytes (defined by previous entry), field name 2 byte numeric, field length in bytes end of repeating block
Data section. Repeat to end of file: 1 byte "deleted" flag. 0 implies valid record, 1 implies deleted record Record containing fields in order specified in schema section, no separators between fields, each field fixed length at maximum specified in schema information
End of file
All numeric values are stored in the header information use the formats of the DataInputStream and DataOutputStream classes. All text values, and all fields (which are text only), contain only 8 bit characters, null terminated if less than the maximum length for the field. The character encoding is 8 bit US ASCII.

**Database schema**
The database that URLyBird uses contains the following fields:
Field descriptive name Database field name Field length Detailed description
Hotel Name name 64 The name of the hotel this vacancy record relates to
City location 64 The location of this hotel
Maximum occupancy of this room size 4 The maximum number of people permitted in this room, not

including infants
Is the room smoking or non-smoking smoking 1 Flag indicating if smoking is permitted. Valid values are "Y" indicating a smoking room, and "N" indicating a non-smoking room
Price per night rate 8 Charge per night for the room. This field includes the currency symbol
Date available date 10 The single night to which this record relates, format is yyyy/mm/dd.
Customer holding this record owner 8 The id value (an 8 digit number) of the customer who has booked this. Note that for this application, you should assume that customers and CSRs know their customer ids. The system you are writing does not interact with these numbers, rather it simply records them. If this field is all blanks, the record is available for sale.

**Deliverables**
**Target Platform and Execution**
Throughout this exercise, you must use exclusively the Java 2 platform. You may develop your code using any implementation of the Java 2 platform, but the submission that you return must have been tested and shown to work under a production (not development) version of the Sun Microsystems' Java 2 platform and that platform must not have been superseded by a new production version for more than 18 months by the time you make your submission.
You are permitted to use any IDE tool you choose, but you must not submit any code that is not your own work. The final program must have no dependencies on any libraries other than those of the Java 2 Platform.
When you submit your assignment, each part (client and server) must be executable using a command of this exact form:
java -jar []
Your programs must not require use of command line arguments other than the single mode flag, which must be supported. Your programs must not require use of command line property specifications. All configuration must be done via a GUI, and must be persistent between runs of the program. Such configuration information must be stored in a file called suncertify.properties which must be located in the current working directory.
The mode flag must be either "server", indicating the server program must run, "alone", indicating standalone mode, or left out entirely, in which case the network client and gui must run.
You must not require manual editing of any files by the examiners.
Packaging of Submissions
All elements of your submission must be packaged in a single JAR file. The JAR file must have the following layout and contents in its root:

- The executable JAR containing the programs. This must be called runme.jar.
- The original, unchanged database file that was supplied to you. Note that you must keep a copy of the original database file supplied to you, and this must be the file you submit. The marking process will expect the exact same data without any changes.
- A directory called code, containing all the source code and related parts of your project. You must create subdirectories within this to reflect your package structure and distribute your source files within those directories.
- A file called version.txt. This must contain pure ASCII (not a word processor format) indicating the exact version of JDK you used, and the host platform you worked on.
- A directory called docs, containing the following items at the top level:
  - These instructions.
  - A subdirectory called javadoc, containing HTML/Javadoc documentation for all classes and interfaces you are submitting.
  - A file called choices.txt that containing pure ASCII (not a word processor format) text describing the significant design choices you made. Detail the problems you perceived, the issues surrounding them, your value judgments, and the decisions that you made. This document should also describe any uncertainties you had regarding the project, and the decisions you made when resolving them.
  - User documentation for the database server and the gui client. If your user documentation is online then you may omit this file. However, if the documentation is not online, you must provide either a single plain ASCII (not word processor format) text document, which must be

called userguide.txt, or multiple HTML files which must all be accessible from a starting point document that must be called userguide.html.

**Marking Process**

The marking is done in three phases. First, software checks that overall structure and nomenclature conform to specification. Second the examiner runs the code ensuring that it functions correctly through the specified operations. If any automatic failures are noted at this stage, the marking process terminates and the assignment is failed.

Provided the essential behavioral requirements of the assignment have been correctly implemented, the examiner proceeds to investigate the design and implementation of your assignment. This process is time consuming, and it is because this is done carefully and thoroughly that submissions take time to grade. The grading process is closely controlled to ensure consistency and fairness, and it is performed according to criteria detailed in the next section. At any time during this process, if an automatic failure is noted, the marking process terminates, and the assignment is failed. For any design choice concerning topics not specifically described in the requirements, marks are awarded for a clear and consistent approach, rather than for any particular solution. Design decisions must be described briefly but clearly in your comments.

In addition to the submission, you will be required to take a written examination. This exam tests your understanding of your submission and asks you to justify a number of design choices embodied in that submission.

Automatic Failures

Where this document uses the word "must" an absolute requirement is being described. If you fail to adhere to such a requirement, your assignment will be failed automatically, and without further evaluation.

**Marking Criteria**

Your work will be evaluated based on the following criteria.

The minimum passing score is 320 out of a possible 400 points.

General Considerations (100 points), Documentation (70 points), Object-oriented design (30 points), User Interface (40 points), Locking (80 points), Data class (40 points), Server (40 points)

When you have completed your assignment and your essay exam, you may submit your assignment to developer-submit_US@oracle.com. Your submission may be made by email. Make sure to include you Prometric ID when you make your submission. You have one year to complete your assignment and your essay. Assessors will take 4-6 weeks to grade our assignment and return results to you.

Next