Java Fundamentals
Chapter 27 Practice Notes

In this chapter we are going to complete the code for our MaterialCatalogDB class - connecting our Lending Library application to the Library database provided with the course materials.

This chapter uses the JDBC objects and methods that we learned about in chapter 26. There is an extra complication however, in that the Material database table stores both books and DVDs, so we need to understand how to tell what type of object a particular material is - whether it's a book or a DVD. I explain how to do this using the *instanceof* operator.

In the video we work through the first 3 stages, together, and then I leave you to have a go at stages 4 and 5 (although of course there is the worked solution both in the video and in the workspaces folder). I've put the full detail of all 5 stages in this PDF however to help you as you work through the video.

Task Overview

Stage 1

Amend the code in our Main method which currently instantiates an instance of the MaterialCatalogMemoryVersion class to instead instantiate an instance of our MaterialCatalogDB class.

Stage 2

Code up the addMaterial method of the MaterialCatalogDBVersion Class

The SQL needed is as follows:

for Books:

```
insert into materials (barcode, title,
author, isbn, numberofpages, branch, type)
values ('abc', 'abc', 'abc', 'abc', 111,
'my branch', 'BOOK')
```

for DVDs:

```
insert into materials (barcode, title,
catalognumber, runningtime, licenced,
branch, type) values ('abc', 'abc', 'abc',
'abc', 'my branch', 111, true, 'DVD').
```

Note that we don't currently store DVD directors in the database!

Stage 3

Code up the getNumberOfMaterials method

We can base this on the retrieveResultSet method we wrote in chapter 26. The SQL needed is:

```
Select count(id) from materials
```

Stage 4

Code up the findMaterial method - this will be similar in concept to the getNumberOfMaterials method.

The SQL needed is:

```
Select * from materials where title like
'%supplied title%'
```

Stage 5

Code up the getMap method. The code will need to loop through all matching materials, create objects from them, and add those objects into a Map. The SQL needed is:

```
Select * from materials
```

This page contains supplementary information to support you in the task. I suggest you watch the video for the first 3 steps, then for steps 4 and 5 try and work out what is needed first, and then use this page as a hint if you get stuck. Don't forget there is a full work through on the video, and sample completed code in the Practicals and Code folder.

Task Detail

Step 1 - amend the Main method to use MaterialCatalogDB

Remember that we previously set up the objects so that both MaterialCatalogMemoryVersion and MaterialCatalogDB implement the MaterialCatalogInterface. We declared the materialCatalog variable in our Main method as an instance of "something which implements this interface" so making this change simply requires:

(1) that we change which class constructor is called and

(2) that we import the correct class into the Main class.

Step 2 - code the addMaterial method

This turns out to be a large and somewhat complex method that brings together database connectivity, interfaces and polymorphism – if you follow the video first time well done, but if not, don't worry – continue with the exercise and then review it again later!

Now let's get onto the detail… there is a complication here – the addMaterial method takes in variables of type Material – these will actually be Books or DVDs. We need to know however whether the item is a Book or DVD as they have different fields. We can find this out by using the instanceof operator – this will allow us to compare our object newMaterial to Book or DVD and we can then cast our newMaterial into an object of type Book or DVD.

Once we have converted our item into a DVD or Book we can:

(1) create a connection to the database

(2) create a prepared statement and bind the parameters

(3) execute the prepared statements

(4) close the objects

We'll need to create try catch blocks, as we did in the previous chapter.

Note that the SQL needed is as follows:

for Books:

```
insert into materials (barcode, title,
author, isbn, numberofpages, branch, type)
values ('abc', 'abc', 'abc', 'abc', 111,
'my branch', 'BOOK')
```

for DVDs*:

```
insert into materials (barcode, title,
catalognumber, runningtime, licenced,
branch, type) values ('abc', 'abc', 'abc',
'abc', 'my branch', 111, true, 'DVD').
```

* You might notice that we don't currently store DVD directors in the database. This was an intentional error – in practice we would store the DVD in the author field, but I didn't wish to include that here as it would add an extra unnecessary complication!

Step 3 - code the getNumberOfMaterials method

Compared to the previous method, this one is quite easy. The steps required are:

(1) create a connection to the database

(2) retrieve a resultset from the database and store the first field of the first row ready to return to the calling method

(3) close the objects

Of course you'll need appropriate try catch blocks.

The SQL needed is:

```
Select count(id) from materials
```

If you want to save time, you can base this on the retrieveResultSet method we wrote in chapter 26…. That's what I do on the video!

Be sure to watch the video!

If you manage to follow the steps and complete the tasks, well done! But it's worth watching the video to ensure your code is similar to mine... and there are some tips in there too!

Step 4- code the findMaterial method

In this method we want to find the first item in the database with a matching title, and return this item to the calling method.

I suggest you start by copying the getNumberOfMaterials method and edit this as needed. The general steps are:

(1) create a connection to the database

(2) retrieve a resultset from the database. The SQL needed is:

```
Select * from materials where title like
'%supplied title%'
```

Here the use of the % symbol before and after the word mean "find all items where this phrase appears anywhere in the field"

(3) Check if the resultset is empty – if it is we will want to throw a MaterialNotFound exception. You can do this check by putting `rs.next()` in an if statement.

(4) If we do have at least 1 item in the resultset, that will contain the data for our book or dvd to return. First, find out whether the item is a DVD or a BOOK by checking the value of the "type" database field. Don't forget to use .equals to compare strings!

(5) Based on the value of the "type" field (which will be either BOOK or DVD in uppercase) we can create an object of the correct type, and in the constructor pass in each of the fields from the resultset. For example, to get the title we might create a local variable and retrieve it's value with a line of code like:

```
String title = rs.getString("title");
```

(6) close the objects

Again, you'll need appropriate try catch blocks.

Step 5- code the getMap method

The code will need to retireve every material from the database, loop through each item, create objects from them, and add those objects into a Map. The map can then be returned to the calling function

A good starting point for this method is to copy the findMaterial method and edit as needed.

The general steps are:

(1) create a map to store the list of books and DVDs – the values of the map should be of type Material. I suggest you use a LinkedHashMap.

(2) create a connection to the database

(3) retrieve a resultset from the database. The SQL needed is:

```
Select * from materials
```

(4) Check if the resultset is empty – if it is we will want to throw an exception… note this shouldn't happen, so a runtime exception is appropriate!

(5) Loop through all the materials. For each one, find out whether the item is a DVD or a BOOK by checking the value of the "type" database field. Don't forget to use .equals to compare strings!

(6) Based on the value of type (which will be either BOOK or DVD) we can create an object of the correct type, and in the constructor pass in each of the fields from the resultset. For example, to get the title we might create a local variable and retrieve it's value with a line of code like:

```
String title = rs.getString("title");
```

(7) Having created each object, add it to the map

(8) Once the loop has finished (we have reached the end of the materials table), close the database objects, and return the map we have been working with to the calling function.

Be sure to watch the video!

If you manage to follow the steps and complete the tasks, well done! But it's worth watching the video to ensure your code is similar to mine... and there are some tips in there too!

If you are struggling with this exercise, don't worry - we're bringing together a number of topics and it's complicated! Make sure you watch the video, and spend some time reviewing the final code to ensure you understand how it works - if you can do that, then you're doing well.