# Assignment 5 6/4/2019

Implement the next release of your term project, employing a database manipulated from your application.

If you already have the skills to set up a database outside of Java, confirm with your facilitator; otherwise, set it up from Java. If you want to use a database other than Derby as described, please check with your facilitator.

The same instructions as previously apply to this completed Word document, the gray text, the 5 page limit, appendices, JUnit tests, and a ReadMe file.

## 5.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from Assignment 4.

This program enables a user to play a card game with the computer. It displays a menu of games from which to choose, Basra being the only one currently viable. (The game of Basra is, apparently, quite popular in Egypt and Lebanon, although this student only just discovered it in a book of family card games this week [The Ultimate Book Of Family Card Games by Oliver Ho].)

The user will be able to select a game, and then will either see a “Not implemented yet; please try again” type of message, or the Basra game will launch.

Launching will include a description of the rules of the game and the set-up.

Rules and set-up will involve a 52-card deck, shuffling of the deck, four cards dealt to each player, four cards initially on the table (“on the floor”), how to choose which card to play, and the capturing and scoring descriptions.

Play will involve the user’s entering of a card value to play, the results of the played card,

a current display of what’s on the floor, and, when it is the computer’s turn, the computer selecting a card out of its hand (via an algorithm).

Since Basra can go on for several rounds, the user can choose to continue or not after a round, whether the game is over or not. Score will be visibly kept. The goal of the game, depending on whose version of the rules you read, is to be the first player to reach 101 points at the end of a round; if there is a tie, play continues until a round results in the tie being broken.

This next release of this project incorporates a database. The database implemented for this project stores “solution sets” of addends for a played card’s value. There are 3 tables in this database. The solutionTable holds 106 solution sets; the cardValueTable holds values of cards that could capture a [solution] set; and the cardKeyTable is the key between these two other tables, which consist only of two attributes that form a unique composite PK, each a FK to one of the other tables, and thereby disallowing multi-valued attributes.

This release also, as a result of the database, allowed this student to come up with an algorithm with which to solve the problem of a card capturing other cards that ADD UP TO its value. **Profuse** amounts of code were added, and **mostly** (**but not completely**) **debugged**.

## 5.2 KEY REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

Title and one or two sentences per requirement. Repeat requirements implemented for prior assignments if they are necessary to provide context. Make it clear which requirements are new vs. old.

### 5.2.1 Database ( NEW)

As described in 5.1, the database implements solution sets as addends for playing cards.

The database appears to work flawlessly, and the queries do what they are supposed to do.

IN ORDER TO RUN MY CODE ON YOUR SYSTEM, please:

1. Go into my Global interface,
2. Edit the url path to your own, and
3. Run the Connect class with its own Main() method.
4. Then you should be all set to run CardGames from its Main() method after that.

### 5.2.2 Capturing cards by: (OLD/NEW)

1. “Sweeping” the floor with a Jack or the 7/Diamonds,
2. Matching the card being played, or
3. Adding the values of cards on the floor to equal a 2, 3,…or 10 that is played.

Numbers 1) and 2) above are old; 3) is brand new (and still has a few bugs).

## 5.3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

Provide an example of input / output showing the new features of your application.

<Your response replaces this>

### Input File(s)

A file I called MyDB is created, written to, and then read from.

### Input / Output

Console I/O:

The console output pertaining to the new capturing feature also contains extraneous output for verification and debugging. (This extraneous output includes data extracted from the database, which is good for observing that the database works, but will not be displayed when the project is completed.) Here is an example:

As you can see, the floor contains a 6 and a 4. I attempted to capture them with my 10/Spades. The solution set for a 10 attempting to capture a 4 and a 6 would look like this: 000101000. (From left to right, each digit in the array represents card values from ace through ten.) The code extracted this solution set (#105) from the database and applied it to remove the 6 and the 4! Unfortunately, the code neglected to also discard the 10/Spades, instead of playing it to the floor.

On the floor: 1) 6/Hearts 2) 2/Clubs 3) 9/Clubs 4) 4/Clubs 5) 7/Diamonds

In your hand: 1) Jack/Diamonds 2) Ace/Hearts 3) 10/Spades 4) 9/Diamonds

Please choose the number of the card you want to play: 3

CARD CHOSEN BY Tini : 10/Spades

cardValueID = 9

Soln #76: 4 aces, 3 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #77: 4 aces, 1 twos, 0 threes, 1 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #78: 4 aces, 0 twos, 2 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #79: 4 aces, 0 twos, 0 threes, 0 fours, 0 fives, 1 sixes, 0 sevens, 0 eights, 0 nines.

Soln #80: 3 aces, 2 twos, 1 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #81: 3 aces, 1 twos, 0 threes, 0 fours, 1 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #82: 3 aces, 0 twos, 1 threes, 1 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #83: 2 aces, 4 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #84: 2 aces, 2 twos, 0 threes, 1 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #85: 2 aces, 1 twos, 2 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #86: 2 aces, 1 twos, 0 threes, 0 fours, 0 fives, 1 sixes, 0 sevens, 0 eights, 0 nines.

Soln #87: 2 aces, 0 twos, 1 threes, 0 fours, 1 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #88: 2 aces, 0 twos, 0 threes, 2 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #89: 2 aces, 0 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 1 eights, 0 nines.

Soln #90: 1 aces, 3 twos, 1 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #91: 1 aces, 2 twos, 0 threes, 0 fours, 1 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #92: 1 aces, 1 twos, 1 threes, 1 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #93: 1 aces, 1 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 1 sevens, 0 eights, 0 nines.

Soln #94: 1 aces, 0 twos, 3 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #95: 1 aces, 0 twos, 1 threes, 0 fours, 0 fives, 1 sixes, 0 sevens, 0 eights, 0 nines.

Soln #96: 1 aces, 0 twos, 0 threes, 1 fours, 1 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #97: 1 aces, 0 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 1 nines.

Soln #98: 0 aces, 3 twos, 0 threes, 1 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #99: 0 aces, 2 twos, 2 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #100: 0 aces, 2 twos, 0 threes, 0 fours, 0 fives, 1 sixes, 0 sevens, 0 eights, 0 nines.

Soln #101: 0 aces, 1 twos, 1 threes, 0 fours, 1 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #102: 0 aces, 1 twos, 0 threes, 2 fours, 0 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

Soln #103: 0 aces, 1 twos, 0 threes, 0 fours, 0 fives, 0 sixes, 0 sevens, 1 eights, 0 nines.

Soln #104: 0 aces, 0 twos, 1 threes, 0 fours, 0 fives, 0 sixes, 1 sevens, 0 eights, 0 nines.

Soln #105: 0 aces, 0 twos, 0 threes, 1 fours, 0 fives, 1 sixes, 0 sevens, 0 eights, 0 nines.

Solution # 105 was found!

Soln #106: 0 aces, 0 twos, 0 threes, 0 fours, 2 fives, 0 sixes, 0 sevens, 0 eights, 0 nines.

solution after extraction from temp = 000101000floorCardValues = 010101101

solutionSet = 000101000

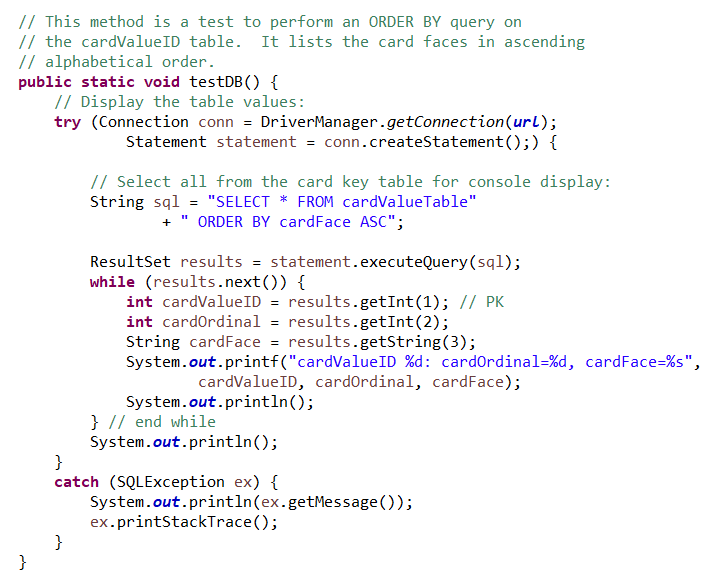
Nano has 3 cards in hand.

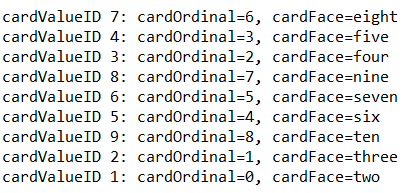
On the floor: 1) 2/Clubs 2) 9/Clubs 3) 7/Diamonds 4) 10/Spades

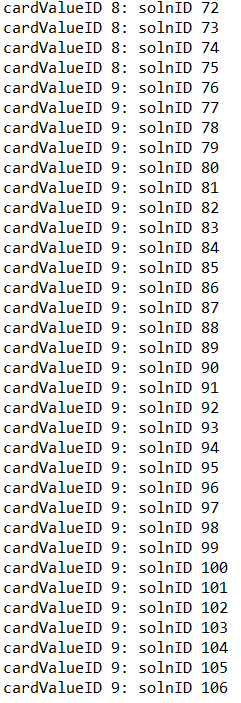
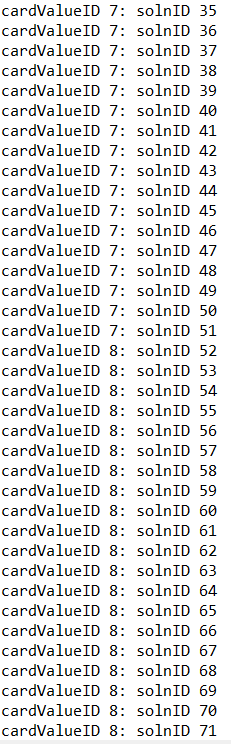
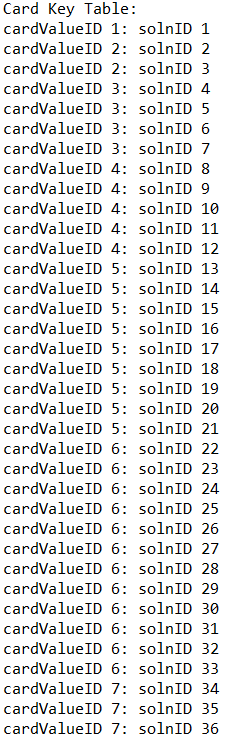
In your hand: 1) Jack/Diamonds 2) Ace/Hearts 3) 9/Diamonds

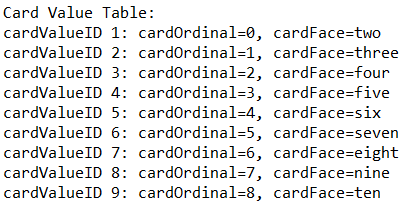
### Output File(s)

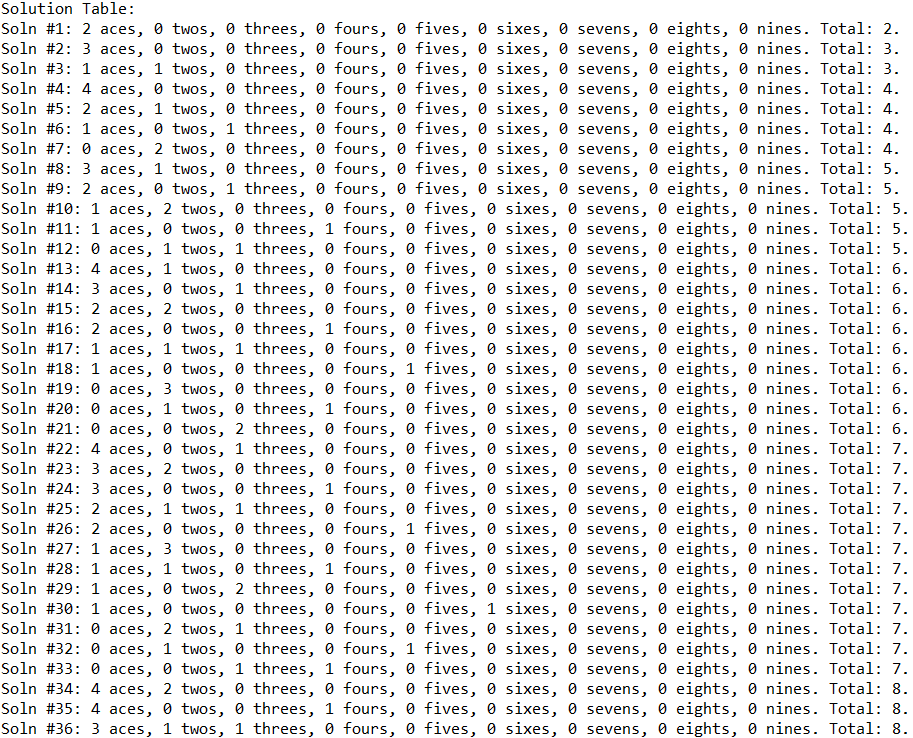
Here is some database output. Displayed first is the output involving an ORDER BY query, and then outputs from SELECT \* queries on the 3 tables.

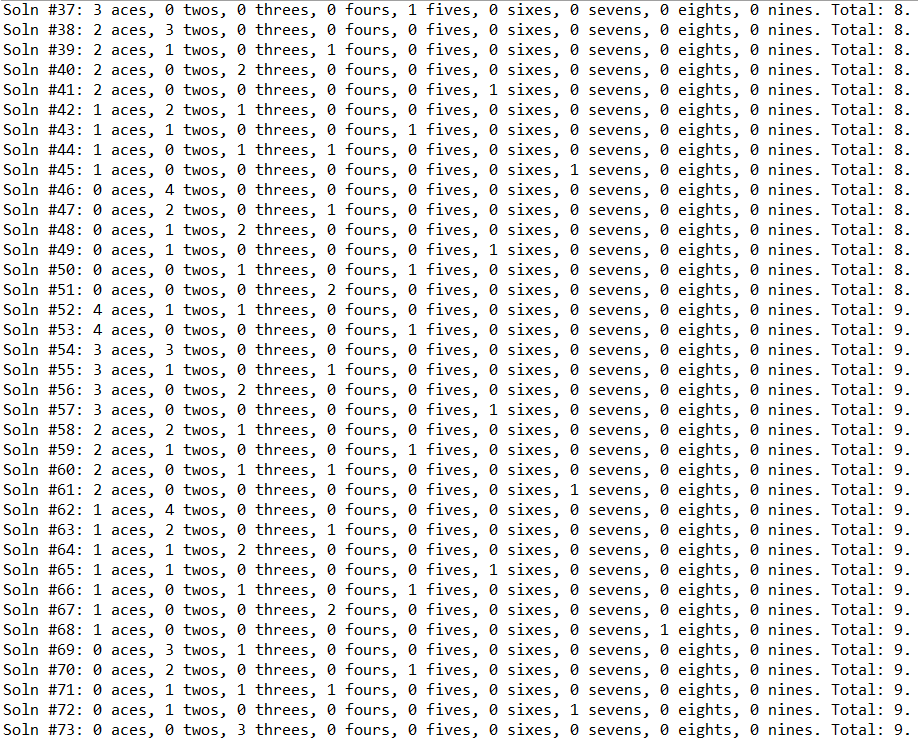


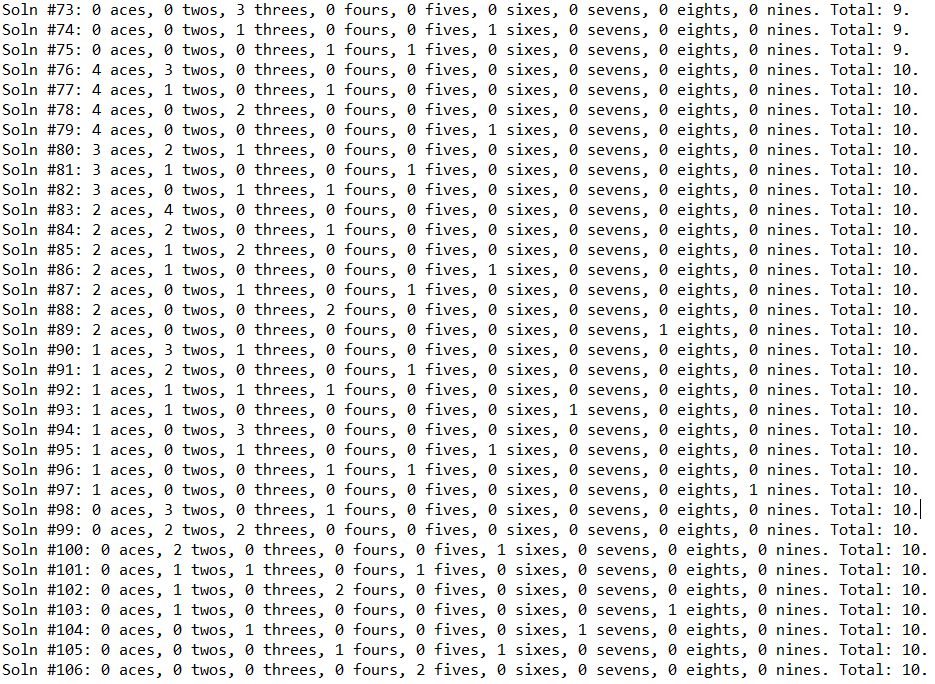








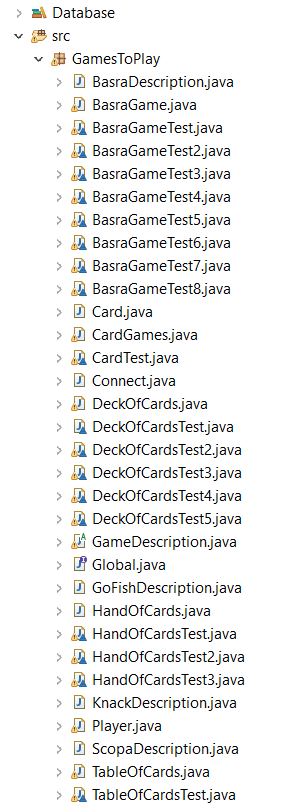


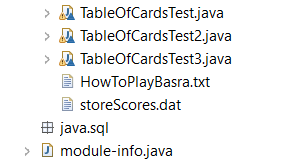


## 5.4 YOUR DIRECTORY

Show a screenshot of your directory. Include all relevant files. This should include JUnit tests.

Although one method was added to BasraGame (checkAddSolutions()) and 10 methods to Connect, each of these involve database I/O, so these were not JUnit-tested. (Also, one method was renamed from “postCapture” to “capturePlayingCard.”)





## 5.5 DESIGN

### 5.5.1 Class Model, Use Case, and Sequence Diagram

## Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red where you applied the features listed below.

Database class indicated and use case (step vi) updated

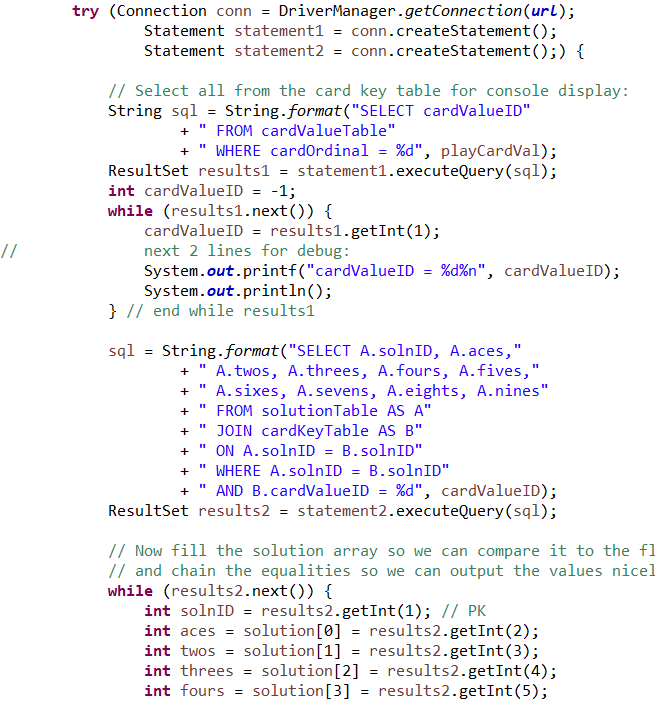
RUML:

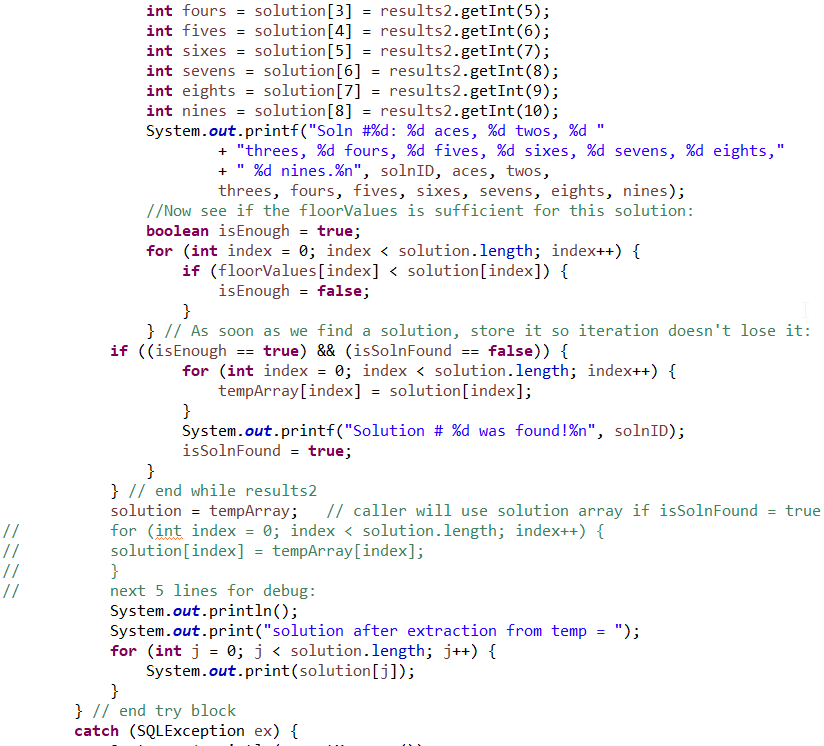
<https://docs.google.com/spreadsheets/d/1qpbNovQZMU1dKEYgk1m8wPgPkSpxfbAxhAMnEjEvAHs/edit?usp=sharing>

(Also, please see Appendix A for ERD.)

### 5.5.2 Code of any kind showing table creation

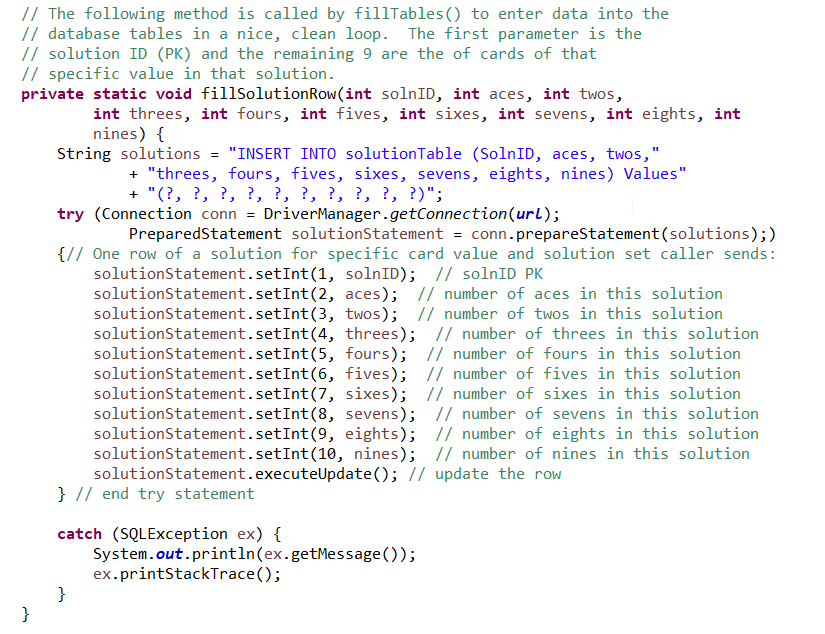
(The following is the code that produced the output in section 5.3, pages 2-4. It comes from checkAddSolutions() in BasraGame, and it is called by capture().):





### 5.5.3 Java code involving the insertion of data

This code is called 106 times in a loop from fillTables() in the Connect class.



### 5.5.4 Java code involving the selection of records, and ordering with SQL

See page 4, under “Output Files.” This code resides in the testDB() method, in the Connect class.

### 5.5.5 Java code involving selection involving at least two tables

See page 9, Section 5.5.2. This code resides in checkAddSolutions(), in the BasraGame class.

## 5.7 YOUR CODE

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to the Blackboard submission. Please contact your facilitator in advance if you want to request an exception.

<Your response here>

## 5.8 Evaluation



# Appendix A: ER Diagram

