```
 1
 2 package cecs323.jdbcproject;
 3
 4 import java.sql.*;
13
14 /**
15  * <h1>CECS323JDBCProject</h1> This is program is designed to be operate in
16  * conjunction with a database of books, publisher, and writing group.
17  *
18  * @author Sotheanith Sok
19  * @version 1.5
20  * @since 03-16-2017
21  */
22 public class CECS323JDBCProject {
23
24     public static void main(String[] args) {
25         // TODO code application logic here
26         Scanner in = new Scanner(System.in);
27         // Input data if required
28         String DBNAME = "JDBCProjectDatabase";
29         String USER = "IAmNotARobot";
30         String PASS = "IAmNotARobot";
31
32         // Database URL
33         String DB_URL = "jdbc:derby://localhost:1527/" + DBNAME + ";user=" + USER +
   ";password=" + PASS;
34         Connection conn = null;
35         // Statement stmt=null;
36         boolean done = false;
37         try {
38             // Register JDBC driver
39             Class.forName("org.apache.derby.jdbc.ClientDriver").newInstance();
40             // Open Connection
41             System.out.println("Connecting to database...");
42             conn = DriverManager.getConnection(DB_URL);
43
44             // Create Datebase object
45             DatabaseOperations d = OpsImplFactory.getOperationsImpl(conn);
46             do {
47                 int choice = menu(in);
48                 switch (choice) {
49                 case 1:
50                     System.out.println("-Listing All Writing Groups-");
51                     listAllWritingGroups(d);
52                     break;
53                 case 2:
54                     System.out.println("-Listing all data for a writing group-");
55                     listDataForAWritingGroup(d, in);
56                     break;
57                 case 3:
58                     System.out.println("-Listing all publishers-");
59                     listAllPublisher(d);
60                     break;
61                 case 4:
62                     System.out.println("-Listing all the data for publisher-");
63                     printDataForAPublisher(d, in);
64                     break;
```

```
65                  case 5:
66                      System.out.println("-Listing all book titles-");
67                      listAllBookTitle(d);
68                      break;
69                  case 6:
70                      System.out.println("-Listing all the data for a book-");
71                      listDataForABook(d, in);
72                      break;
73                  case 7:
74                      System.out.println("-Inserting a new book-");
75                      insertABook(d, in);
76                      break;
77                  case 8:
78                      System.out.println("-Inserting a new publisher-");
79                      insertAPublisher(d, in);
80                      break;
81                  case 9:
82                      System.out.println("-Removing a book-");
83                      removeABook(d, in);
84                      break;
85                  case 10:
86                      System.out.println("-Exiting-");
87                      done = true;
88                      break;
89                  default:
90                      System.out.println("-Invalid Input-");
91                      break;
92                  }
93              } while (!done);
94              // Close resource when there isn't any error.
95              conn.close();
96          } catch (SQLException se) {
97              System.out.println("ERROR: Connection to Database failed!!!");
98          } catch (Exception e) {
99              // Testing for unexpected exception.
100             System.out.println("Unknown Exception was threw to main");
101             System.out.println(e);
102         } finally {
103             // Error caused by closing resources.
104             try {
105                 conn.close();
106                 in.close();
107             } catch (SQLException se) {
108                 System.out.println("H3");
109                 se.printStackTrace();
110                 // This error is mostly caused by wrong database info.
111             } catch (NullPointerException np) {
112                 System.out.println("ERROR: Database related informations are incorrect.");
113             }
114         }
115     }
116
117     /**
118      * Print menu and get user input
119      *
120      * @param in Scanner for keyboard
121      * @return valid user choice
```

```
122      */
123     public static int menu(Scanner in) {
124         boolean done = false;
125         int choice = 0;
126         System.out.println("--Menu--");
127         System.out.println("1. List all writing groups ");
128         System.out.println("2. List all data for a writing group");
129         System.out.println("3. List all publishers");
130         System.out.println("4. List all the data for a publisher");
131         System.out.println("5. List all book titles");
132         System.out.println("6. List all the data for a book");
133         System.out.println("7. Insert a new book");
134         System.out.println("8. Insert a new publisher");
135         System.out.println("9. Remove a book");
136         System.out.println("10. Exit");
137         while (!done) {
138             try {
139                 System.out.print("Enter: ");
140                 choice = in.nextInt();
141                 if (!(choice >= 1 && choice <= 10)) {
142                     throw new NumberFormatException();
143                 }
144                 done = true;
145             } catch (InputMismatchException ime) {
146                 in.next();
147                 System.out.print("Invalid Input. Re-enter: ");
148             } catch (NumberFormatException nfe) {
149                 System.out.print("Invalid Input. Re-enter: ");
150             }
151         }
152         return choice;
153     }
154 
155     /**
156      * List all data related to all writing groups
157      *
158      * @param w WritingGroupOperations object.
159      * @throws SQLException
160      */
161     public static void listAllWritingGroups(DatabaseOperations w) {
162         try {
163             List<WritingGroup> list = w.listWritingGroups();
164             // Check if WritingGroups is empty.
165             if (list.size() == 0) {
166                 throw new SQLException();
167             }
168             // Print WritingGroups.
169             System.out.printf("%-20s%-20s%-20s%-20s\n", "GroupName", "HeadWriter",
    "YearFormed", "Subject");
170             for (int i = 0; i < list.size(); i++) {
171                 System.out.printf("%-20s%-20s%-20s%-20s\n", list.get(i).groupName,
    list.get(i).headWriter,
172                         list.get(i).subject, list.get(i).yearFormed);
173             }
174         } catch (SQLException s) {
175             System.out.println("ERROR: WritingGroups is empty.");
176         }
```

```
177     }
178
179     /**
180      * List all data for a specific WritingGroup (4)
181      *
182      * @param w WritingGroupOperations object
183      * @param in Scanner for keyboard
184      * @throws SQLException
185      */
186     public static void listDataForAWritingGroup(DatabaseOperations w, Scanner in) {
187         try {
188             List<String> list = w.listWritingGroupNames();
189             // Check if WritingGroups is empty.
190             if (list.size() == 0) {
191                 throw new SQLException();
192             }
193             // Print available WritingGroups.
194             System.out.println("-Available Group-");
195             for (int i = 0; i < list.size(); i++) {
196                 System.out.println(list.get(i));
197             }
198             // Get input.
199             System.out.print("Enter group name: ");
200             String groupName = in.next();
201             WritingGroup k = w.getWritingGroup(groupName);
202             // Print result
203             if (k.groupName != null) {
204                 System.out.printf("%-20s%-20s%-20s%-20s\n", "GroupName", "HeadWriter",
    "YearFormed", "Subject");
205                 System.out.printf("%-20s%-20s%-20s%-20s\n", k.groupName, k.headWriter,
    k.yearFormed, k.subject);
206             }
207         } catch (SQLException s) {
208             System.out.println("ERROR: WritingGroups is empty.");
209         } catch (NullPointerException np) {
210             System.out.println("ERROR: WritingGroup was not found.");
211         }
212     }
213
214     /**
215      * List information related to all publishers (4)
216      *
217      * @param p PublisherOperations object
218      * @throws SQLException
219      */
220     public static void listAllPublisher(DatabaseOperations p) {
221         try {
222             List<Publisher> list = p.listPublishers();
223             // Check if publishers is empty.
224             if (list.size() == 0) {
225                 throw new SQLException();
226             }
227             // Print result.
228             System.out.printf("%-20s%-30s%-20s%-20s\n", "PublisherName", "PublisherAddress",
    "PublisherPhone",
229                     "PublisherEmail");
230             for (int i = 0; i < list.size(); i++) {
```

```
231                 System.out.printf("%-20s%-30s%-20s%-20s\n", list.get(i).publisherName,
    list.get(i).publisherAddress,
232                         list.get(i).publisherPhone, list.get(i).publisherEmail);
233             }
234         } catch (SQLException s) {
235             System.out.println("ERROR: Publishers is empty.");
236         }
237     }
238 
239     /**
240      * Print all data for a publisher
241      *
242      * @param d DatabaseOperations object
243      */
244     public static void printDataForAPublisher(DatabaseOperations d, Scanner in) {
245         try {
246             List<String> list = d.listPublisherNames();
247             // Check if Publishers is empty.
248             if (list.size() == 0) {
249                 throw new SQLException();
250             }
251             // Print available publishers
252             System.out.println("-Available Publishers-");
253             for (int i = 0; i < list.size(); i++) {
254                 System.out.println(list.get(i));
255             }
256             // Get input
257             System.out.print("Enter publisher name: ");
258             in.nextLine();
259             String pubName = in.nextLine();
260             Publisher p = d.getPublisher(pubName);
261             // Print result
262             if (p.publisherName != null) {
263                 System.out.printf("%-20s%-30s%-20s%-20s\n", "PublisherName",
    "PublisherAddress", "PublisherPhone",
264                         "PublisherEmail");
265                 System.out.printf("%-20s%-30s%-20s%-20s\n", p.publisherName,
    p.publisherAddress, p.publisherPhone,
266                         p.publisherEmail);
267             }
268         } catch (SQLException e) {
269             System.out.println("ERROR: Publishers is empty.");
270         } catch (NullPointerException np) {
271             System.out.println("ERROR: Publisher was not found.");
272         }
273     }
274 
275     /**
276      * List all the book title
277      *
278      * @param b
279      * @throws SQLException
280      */
281     public static void listAllBookTitle(DatabaseOperations b) {
282         try {
283             List<String> list = b.listBookTitles();
284             // Check if Books is empty.
```

```
285             if (list.size() == 0) {
286                 throw new SQLException();
287             }
288             // Print result.
289             System.out.printf("%-10s", "BookTitle\n");
290             for (int i = 0; i < list.size(); i++) {
291                 System.out.printf("%-10s\n", list.get(i));
292             }
293         } catch (SQLException s) {
294             System.out.println("ERROR: Books is empty.");
295         }
296     }
297 
298     /**
299      * List all data for a specific book including related writing group and
300      * publisher.
301      *
302      * @param b BookOperations object
303      * @param in Scanner for keyboard
304      * @throws SQLException
305      */
306     public static void listDataForABook(DatabaseOperations b, Scanner in) {
307         try {
308             // Check if Books is empty.
309             if (b.listBookTitles().size() == 0) {
310                 throw new SQLException();
311             }
312             // Print available Publishers and WritingGroups.
313             printAvailabeBooks(b);
314             // Get input.
315             System.out.print("Enter BookTitle:");
316             in.nextLine();
317             String bookTitle = in.nextLine();
318             System.out.print("Enter groupName: ");
319             String groupName = in.nextLine();
320             Book book = b.getBook(new BookKeyData(bookTitle, groupName));
321             BookDetail bookDetail = b.getBookDetails(new BookKeyData(bookTitle, groupName));
322             // Print result.
323             if (book.groupName != null) {
324                 System.out.printf("%-40s%-20s%-20s%-20s%-20s%-20s%-20s%-30s%-30s%-20s\n",
    "BookTitle",
325                         "YearPublished", "NumberPages", "GroupName", "HeadWriter",
    "YearFormed", "Subject",
326                         "PublisherName", "PublisherAddress", "PublisherPhone",
    "PublisherEmail");
327                 System.out.printf("%-40s%-20s%-20s%-20s%-20s%-20s%-20s%-30s%-30s%-20s\n",
    book.bookTitle,
328                         book.yearPublished, book.numberPages,
    bookDetail.writingGroup.groupName,
329                         bookDetail.writingGroup.headWriter,
    bookDetail.writingGroup.yearFormed,
330                         bookDetail.writingGroup.subject, bookDetail.publisher.publisherName,
331                         bookDetail.publisher.publisherAddress,
    bookDetail.publisher.publisherPhone,
332                         bookDetail.publisher.publisherEmail);
333             }
334         } catch (SQLException s) {
```

```
335             System.out.println("ERROR: Books is empty.");
336         } catch (NullPointerException np) {
337             System.out.println("ERROR: Book was not found.");
338         }
339     }
340 
341     /**
342      * Insert a new book into database
343      *
344      * @param d DatabaseOpeartions object
345      * @param in Scanner for keyboard
346      * @throws SQLIntegrityConstraintViolationException
347      * @throws SQLException
348      */
349     public static void insertABook(DatabaseOperations d, Scanner in) {
350         try {
351             if (d.listWritingGroupNames().size() == 0 || d.listPublisherNames().size() == 0) {
352                 throw new SQLException();
353             }
354             // Print out available publisher and group.
355             System.out.println("-Avaialbe Publishers-");
356             List<String> s = d.listPublisherNames();
357             for (int i = 0; i < s.size(); i++) {
358                 System.out.println(s.get(i));
359             }
360             System.out.println("-Avaialbe WritingGroups-");
361             s = d.listWritingGroupNames();
362             for (int i = 0; i < s.size(); i++) {
363                 System.out.println(s.get(i));
364             }
365             // Get input.
366             System.out.print("Enter BookTitle: ");
367             in.nextLine();
368             String bookTitle = in.nextLine();
369             System.out.print("Enter YearPublished: ");
370             String yearPublished = in.nextLine();
371             System.out.print("Enter NumberPages: ");
372             int numberPages = in.nextInt();
373             System.out.print("Enter GroupName: ");
374             in.nextLine();
375             String groupName = in.nextLine();
376             System.out.print("Enter PublisherName: ");
377             String publisherName = in.nextLine();
378             // Insert into Books.
379             d.insertBook(new Book(bookTitle, groupName, publisherName, yearPublished,
    numberPages));
380         } catch (SQLException s) {
381             System.out.println("ERROR: Unable to insert book when publishers or writing groups
    is empty.");
382         } catch (IllegalArgumentException iae) {
383             System.out.println("ERROR: YearPublished should be integer. Insertion Fail!!!");
384         } catch (InputMismatchException im) {
385             System.out.println("ERROR: NumberPages should be integer. Insertion Fail!!!");
386         }
387     }
388 
389     /**
```

```
390      * Insert and replace old publisher with a new one
391      *
392      * @param d DatabaseOperations objects
393      * @param in Scanner for keyboard
394      * @throws SQLIntegrityConstraintViolationException
395      * @throws SQLException
396      */
397     public static void insertAPublisher(DatabaseOperations d, Scanner in) {
398         try {
399             // Check if Publishers is empty.
400             if (d.listPublisherNames().size() == 0) {
401                 throw new SQLException();
402             }
403             // Print available publishers
404             System.out.println("-Avaialbe Publishers-");
405             List<String> s = d.listPublisherNames();
406             for (int i = 0; i < s.size(); i++) {
407                 System.out.println(s.get(i));
408             }
409             // Get input.
410             System.out.print("Enter OldPublisherName: ");
411             in.nextLine();
412             String oldPub = in.nextLine();
413             System.out.println("-Get New Publisher Info-");
414             System.out.print("Enter PublisherName: ");
415             String publisherName = in.nextLine();
416             System.out.print("Enter PublisherAddress: ");
417             String publisherAddress = in.nextLine();
418             System.out.print("Enter PublisherPhone: ");
419             String publisherPhone = in.nextLine();
420             System.out.print("Enter PublisherEmail: ");
421             String publisherEmail = in.nextLine();
422             // Check if oldPublisher actually exist.
423             if (d.getPublisher(oldPub).publisherName == null) {
424             }
425             // Replace publisher
426             d.insertPublisher(new Publisher(publisherName, publisherAddress, publisherPhone,
    publisherEmail));
427             d.replacePublisher(oldPub, publisherName);
428             d.deletePublisher(oldPub);
429         } catch (NullPointerException np) {
430             System.out.println("ERROR: Old publisher was not found. Insertion Fail!!!");
431         } catch (SQLException s) {
432             System.out.println("ERROR: Publisher is empty");
433         }
434     }
435
436     /**
437      * Remove book based on title and group name
438      *
439      * @param d DatabaseOperations object
440      * @param in Scanner for keyboard
441      * @throws SQLException
442      */
443     public static void removeABook(DatabaseOperations d, Scanner in) {
444         try {
445             // Check if Books is empty.
```

```
446             if (d.listBookTitles().size() == 0) {
447                 throw new SQLException();
448             }
449             // Print available Publishers and WritingGroups
450             printAvailabeBooks(d);
451             // Get input
452             System.out.print("Enter BookTitle:");
453             in.nextLine();
454             String bookTitle = in.nextLine();
455             System.out.print("Enter groupName: ");
456             String groupName = in.nextLine();
457             // Check if the book existed.
458             if (d.getBook(bookTitle, groupName).bookTitle == null) {
459                 throw new NullPointerException();
460             }
461             // Perform operation
462             d.deleteBook(new BookKeyData(bookTitle, groupName));
463         } catch (SQLException s) {
464             System.out.println("ERROR: Books is empty.");
465         } catch (NullPointerException iie) {
466             System.out.println("ERROR: Book doesn't not existed in the database.");
467         }
468     }
469 
470     /**
471      * This function is used to print available bookTitle and groupName
472      * combination.
473      *
474      * @param d DatabaseOperations object
475      * @throws SQLException the exception which will be handle by other
476      *             functions.
477      */
478     public static void printAvailabeBooks(DatabaseOperations d) throws SQLException {
479         List<Book> list = d.listBooks();
480         System.out.println("-Avaialbe Book-");
481         System.out.printf("%-40s%-10s\n", "BookTitle", "GroupName");
482         for (int i = 0; i < list.size(); i++) {
483             System.out.printf("%-40s%-10s\n", list.get(i).bookTitle, list.get(i).groupName);
484         }
485     }
486 }
487 
```

```java
package cecs323.jdbcproject.interconnect;

import cecs323.jdbcproject.pojos.Book;

/**
 * The DatabaseOperations interface defines the interface for a class that can
 * perform operations on the database.
 *
 * @author Nicholas Utz
 */
public interface DatabaseOperations {
    /**
     * Returns a {@link List} of Strings, containing the titles of all of the
     * entries in the Books table.
     *
     * @return list of book titles
     */
    public List<String> listBookTitles() throws SQLException;

    /**
     * Returns a {@link List} of {@link Book}s, containing all of the
     * information in the Books table.
     *
     * @return list of all books
     */
    public List<Book> listBooks() throws SQLException;

    /**
     * Returns a {@link Book} storing all of the data pertaining to the book
     * with the given title, written by the WritingGroup with the given name.
     *
     * @param title the title of the book to fetch
     * @param writingGroup the writing group that wrote the book to fetch
     * @return book info
     */
    public Book getBook(String title, String writingGroup) throws SQLException;

    /**
     * Returns a {@link Book} storing all of the data pertaining to the book
     * with the primary key data.
     *
     * @param key the key data of the book to fetch
     * @return book info
     */
    public Book getBook(BookKeyData key) throws SQLException;

    /**
     * Returns a {@link BookDetail} object, containing all available data
     * pertaining to the book with the given title, written by the given writing
     * group, including the publisher and writing group.
     *
     * @param title the title of the book to fetch
     * @param writingGroup the writing group name of the book to fetch
     * @return book details
     */
    public BookDetail getBookDetails(String title, String writingGroup) throws SQLException;

```

```java
    /**
     * Returns a {@link BookDetail} object, storing all of the data pertaining
     * to the book with the given primary key data, and the publisher and
     * writing group of the book.
     *
     * @param key the key of the book to fetch
     * @return book details
     */
    public BookDetail getBookDetails(BookKeyData key) throws SQLException;

    /**
     * Inserts the given book into the books table.
     *
     * @param book the book to insert
     * @throws SQLException if a SQLException occurs while attempting to insert
     */
    public void insertBook(Book book) throws SQLIntegrityConstraintViolationException,
  SQLException;

    /**
     * Replaces the given old publisher name with a new one, for all books
     * published by the old publisher.
     *
     * @param oldName the name of the publisher to be replaced
     * @param newName the name of the publisher replacing the old one
     * @throws SQLException if a SQLException occurs while updating
     */
    public void replacePublisher(String oldName, String newName)
            throws SQLIntegrityConstraintViolationException, SQLException;

    /**
     * Deletes the book with the given title and writing group from the books
     * table.
     *
     * @param title the title of the book to delete
     * @param writingGroup the writing group who wrote the book to delete
     * @throws SQLException if a SQLException occurs while deleting
     */
    public void deleteBook(String title, String writingGroup) throws SQLException;

    /**
     * Deletes the book with the given primary key data from the books table.
     *
     * @param key the primary key data of the book to delete
     * @throws SQLException if a SQL exception occurs while deleting
     */
    public void deleteBook(BookKeyData key) throws SQLException;

    /**
     * Returns a {@link List} of {@link String}s, representing the names of all
     * of the entries in the Publishers table.
     *
     * @return list of publisher names
     */
    public List<String> listPublisherNames() throws SQLException;

    /**
```

```
121      * Returns a {@link List} of {@link Publisher}s, representing all of the
122      * data in the Publishers table.
123      *
124      * @return list of publishers
125      */
126     public List<Publisher> listPublishers() throws SQLException;
127 
128     /**
129      * Returns a {@link Publisher} object, storing all the data stored for the
130      * publisher with the given name in the Publishers table.
131      *
132      * @param name the name of the publisher to fetch
133      * @return publisher info
134      */
135     public Publisher getPublisher(String name) throws SQLException;
136 
137     /**
138      * Inserts a new entry into the Publishers table, using the given
139      * {@link Publisher} as a source of attribute data.
140      *
141      * @param info the info of the publisher to insert
142      * @throws java.sql.SQLException if a SQLException occurs while inserting
143      */
144     public void insertPublisher(Publisher info) throws
    SQLIntegrityConstraintViolationException, SQLException;
145 
146     /**
147      * Deletes the Publisher with the given name from the publishers table.
148      *
149      * @param name the name of the publisher to delete
150      * @throws SQLIntegrityConstraintViolationException if there is a Book
151      *             dependent on the named publisher
152      * @throws SQLException if there is a problem deleting the publisher
153      */
154     public void deletePublisher(String name) throws SQLIntegrityConstraintViolationException,
    SQLException;
155 
156     /**
157      * Returns a {@link List} of Strings, containing the names of all of the
158      * WritingGroups in the WritingGroups table.
159      *
160      * @return list of writing groups' names
161      */
162     public List<String> listWritingGroupNames() throws SQLException;
163 
164     /**
165      * Returns a {@link List} of {@link WritingGroup}s, representing all of the
166      * data in the WritingGroups table.
167      *
168      * @return list of WritingGroups
169      */
170     public List<WritingGroup> listWritingGroups() throws SQLException;
171 
172     /**
173      * Returns a {@link WritingGroup} object containing the data stored in the
174      * WritingGroups table for the WritingGroup with the given name.
175      *
```

```
176      * @param name the name of the WritingGroup to fetch
177      * @return writing group info
178      * @throws NullPointerException if there is no entry in the writing groups
179      *             table with the given name.
180      */
181     public WritingGroup getWritingGroup(String name) throws SQLException;
182 
183     /**
184      * Inserts a row in the WritingGroups table with the attribute values stored
185      * in the given {@link WritingGroup} object.
186      *
187      * @param group the writing group to insert
188      * @throws SQLException if an exception is thrown while trying to insert
189      */
190     public void insertWritingGroup(WritingGroup group) throws
    SQLIntegrityConstraintViolationException, SQLException;
191 
192     /**
193      * Deletes the writing group with the given name.
194      *
195      * @param groupName the name of the writing group to delete
196      * @throws SQLException if an exception is raised while deleting
197      */
198     public void deleteWritingGroup(String groupName) throws
    SQLIntegrityConstraintViolationException, SQLException;
199 }
200 
```

```java
package impl;

import cecs323.jdbcproject.interconnect.DatabaseOperations;

/**
 * The OpsImplFactory class is a factory for classes that implement the
 * *Operations interfaces defined in {@link cecs323.jdbcproject.interconnect}.
 *
 * @author Nicholas Utz
 */
public class OpsImplFactory {

    public static DatabaseOperations getOperationsImpl(Connection con) throws SQLException {
        return new OpsImpl(con);
    }
}

class OpsImpl implements DatabaseOperations {

    private final Connection con;

    private static final String SQL_GET_TITLES = "SELECT booktitle FROM books";
    private static final String SQL_GET_BOOKS = "SELECT * FROM books";
    private static final String SQL_GET_BOOK = "SELECT * FROM books WHERE booktitle=? AND groupname=?";
    private static final String SQL_INSERT_BOOK = "INSERT INTO books (groupname, booktitle, "
            + "publishername, yearpublished, numberpages) VALUES (?,?,?,?,?)";
    private static final String SQL_UPDATE_PUBLISHERS = "UPDATE books SET publishername=? WHERE publishername=?";
    private static final String SQL_DELETE_BOOK = "DELETE FROM books WHERE booktitle=? AND groupname=?";
    private static final String SQL_GET_PUBLISHER_NAMES = "SELECT publishername FROM publishers";
    private static final String SQL_GET_PUBLISHERS = "SELECT * FROM publishers";
    private static final String SQL_GET_PUBLISHER = "SELECT * FROM publishers WHERE publishername=?";
    private static final String SQL_INSERT_PUBLISHER = "INSERT INTO publishers (publishername, "
            + "publisheraddress, publisherphone, publisheremail) VALUES (?, ?, ?, ?)";
    private static final String SQL_DELETE_PUBLISHER = "DELETE FROM publishers WHERE publishername=?";
    private static final String SQL_GET_GROUP_NAMES = "SELECT groupname FROM writinggroups";
    private static final String SQL_GET_GROUPS = "SELECT * FROM writinggroups";
    private static final String SQL_GET_WRITING_GROUP = "SELECT * FROM writinggroups WHERE groupname=?";
    private static final String SQL_INSERT_WRITING_GROUP = "INSERT INTO writinggroups (groupname, "
            + "headwriter, yearformed, subject) VALUES (?,?,?,?)";
    private static final String SQL_DELETE_WRITING_GROUP = "DELETE FROM writinggroups WHERE groupname=?";

    private final PreparedStatement PSTMT_GET_BOOK;
    private final PreparedStatement PSTMT_INSERT_BOOK;
    private final PreparedStatement PSTMT_UPDATE_PUBLISHERS;
    private final PreparedStatement PSTMT_DELETE_BOOK;
    private final PreparedStatement PSTMT_GET_PUBLISHER;
    private final PreparedStatement PSTMT_INSERT_PUBLISHER;
```

```
    private final PreparedStatement PSTMT_DELETE_PUBLISHER;
    private final PreparedStatement PSTMT_GET_WRITING_GROUP;
    private final PreparedStatement PSTMT_INSERT_WRITING_GROUP;
    private final PreparedStatement PSTMT_DELETE_WRITING_GROUP;

    public OpsImpl(Connection con) throws SQLException {
        this.con = con;
        this.PSTMT_GET_BOOK = con.prepareStatement(SQL_GET_BOOK);
        this.PSTMT_INSERT_BOOK = con.prepareStatement(SQL_INSERT_BOOK);
        this.PSTMT_UPDATE_PUBLISHERS = con.prepareStatement(SQL_UPDATE_PUBLISHERS);
        this.PSTMT_DELETE_BOOK = con.prepareStatement(SQL_DELETE_BOOK);
        this.PSTMT_GET_PUBLISHER = con.prepareStatement(SQL_GET_PUBLISHER);
        this.PSTMT_INSERT_PUBLISHER = con.prepareStatement(SQL_INSERT_PUBLISHER);
        this.PSTMT_DELETE_PUBLISHER = con.prepareStatement(SQL_DELETE_PUBLISHER);
        this.PSTMT_GET_WRITING_GROUP = con.prepareStatement(SQL_GET_WRITING_GROUP);
        this.PSTMT_INSERT_WRITING_GROUP = con.prepareStatement(SQL_INSERT_WRITING_GROUP);
        this.PSTMT_DELETE_WRITING_GROUP = con.prepareStatement(SQL_DELETE_WRITING_GROUP);
    }

    @Override
    public List<String> listBookTitles() throws SQLException {
        Statement stmt = this.con.createStatement();
        List<String> titles = new LinkedList<>();
        ResultSet results = stmt.executeQuery(SQL_GET_TITLES);

        while (results.next()) {
            titles.add(results.getString(1));
        }

        results.close();
        stmt.close();

        return titles;
    }

    @Override
    public List<Book> listBooks() throws SQLException {
        Statement stmt = this.con.createStatement();
        List<Book> books = new LinkedList<>();
        ResultSet results = stmt.executeQuery(SQL_GET_BOOKS);

        while (results.next()) {
            books.add(new Book(results.getString(2), results.getString(1),
  results.getString(3), results.getString(4),
                    results.getInt(5)));
        }

        results.close();
        stmt.close();

        return books;
    }

    @Override
    public Book getBook(String title, String writingGroup) throws SQLException {
        this.PSTMT_GET_BOOK.setString(1, title);
        this.PSTMT_GET_BOOK.setString(2, writingGroup);
```

```
117         ResultSet results = this.PSTMT_GET_BOOK.executeQuery();
118 
119         if (results.next()) {
120             Book book = new Book(results.getString(2), results.getString(1),
    results.getString(3), results.getString(4),
121                     results.getInt(5));
122             results.close();
123             return book;
124 
125         } else {
126             results.close();
127             return null;
128         }
129     }
130 
131     @Override
132     public Book getBook(BookKeyData key) throws SQLException {
133         return getBook(key.bookTitle, key.writingGroup);
134     }
135 
136     @Override
137     public BookDetail getBookDetails(String title, String writingGroup) throws SQLException {
138         Book book = getBook(title, writingGroup);
139 
140         if (book == null) {
141             return null;
142         }
143 
144         Publisher pub = getPublisher(book.publisherName);
145         WritingGroup wg = getWritingGroup(writingGroup);
146 
147         if (pub == null || wg == null) {
148             return null;
149         }
150 
151         return new BookDetail(book, pub, wg);
152     }
153 
154     @Override
155     public BookDetail getBookDetails(BookKeyData key) throws SQLException {
156         return getBookDetails(key.bookTitle, key.writingGroup);
157     }
158 
159     @Override
160     public void insertBook(Book book) throws SQLIntegrityConstraintViolationException,
    SQLException {
161         if (!checkYearString(book.yearPublished)) {
162             throw new IllegalArgumentException("Book yearPublished is inproperly formatted");
163         }
164         this.PSTMT_INSERT_BOOK.setString(1, book.groupName);
165         this.PSTMT_INSERT_BOOK.setString(2, book.bookTitle);
166         this.PSTMT_INSERT_BOOK.setString(3, book.publisherName);
167         this.PSTMT_INSERT_BOOK.setString(4, book.yearPublished);
168         this.PSTMT_INSERT_BOOK.setInt(5, book.numberPages);
169 
170         PSTMT_INSERT_BOOK.executeUpdate();
171     }
```

```
172
173     @Override
174     public void replacePublisher(String oldName, String newName)
175             throws SQLIntegrityConstraintViolationException, SQLException {
176         this.PSTMT_UPDATE_PUBLISHERS.setString(1, newName);
177         this.PSTMT_UPDATE_PUBLISHERS.setString(2, oldName);
178
179         this.PSTMT_UPDATE_PUBLISHERS.executeUpdate();
180     }
181
182     @Override
183     public void deleteBook(String title, String writingGroup) throws SQLException {
184         this.PSTMT_DELETE_BOOK.setString(1, title);
185         this.PSTMT_DELETE_BOOK.setString(2, writingGroup);
186         this.PSTMT_DELETE_BOOK.executeUpdate();
187     }
188
189     @Override
190     public void deleteBook(BookKeyData key) throws SQLException {
191         deleteBook(key.bookTitle, key.writingGroup);
192     }
193
194     @Override
195     public List<String> listPublisherNames() throws SQLException {
196         Statement stmt = this.con.createStatement();
197         List<String> pubs = new LinkedList<>();
198         ResultSet results = stmt.executeQuery(SQL_GET_PUBLISHER_NAMES);
199
200         while (results.next()) {
201             pubs.add(results.getString(1));
202         }
203
204         results.close();
205         stmt.close();
206
207         return pubs;
208     }
209
210     @Override
211     public List<Publisher> listPublishers() throws SQLException {
212         Statement stmt = this.con.createStatement();
213         List<Publisher> pubs = new LinkedList<>();
214         ResultSet results = stmt.executeQuery(SQL_GET_PUBLISHERS);
215
216         while (results.next()) {
217             pubs.add(new Publisher(results.getString(1), results.getString(2),
    results.getString(3),
218                     results.getString(4)));
219         }
220
221         return pubs;
222     }
223
224     @Override
225     public Publisher getPublisher(String name) throws SQLException {
226         this.PSTMT_GET_PUBLISHER.setString(1, name);
227         ResultSet results = this.PSTMT_GET_PUBLISHER.executeQuery();
```

```
228
229         if (results.next()) {
230             Publisher pub = new Publisher(results.getString(1), results.getString(2),
    results.getString(3),
231                     results.getString(4));
232             results.close();
233             return pub;
234
235         } else {
236             return null;
237         }
238     }
239
240     @Override
241     public void insertPublisher(Publisher info) throws
    SQLIntegrityConstraintViolationException, SQLException {
242         PSTMT_INSERT_PUBLISHER.setString(1, info.publisherName);
243         PSTMT_INSERT_PUBLISHER.setString(2, info.publisherAddress);
244         PSTMT_INSERT_PUBLISHER.setString(3, info.publisherPhone);
245         PSTMT_INSERT_PUBLISHER.setString(4, info.publisherEmail);
246         PSTMT_INSERT_PUBLISHER.execute();
247     }
248
249     @Override
250     public void deletePublisher(String name) throws SQLIntegrityConstraintViolationException,
    SQLException {
251         PSTMT_DELETE_PUBLISHER.setString(1, name);
252         PSTMT_DELETE_PUBLISHER.execute();
253     }
254
255     @Override
256     public List<String> listWritingGroupNames() throws SQLException {
257         Statement stmt = con.createStatement();
258         List<String> names = new LinkedList<>();
259         ResultSet results = stmt.executeQuery(SQL_GET_GROUP_NAMES);
260
261         while (results.next()) {
262             names.add(results.getString(1));
263         }
264
265         results.close();
266         stmt.close();
267
268         return names;
269     }
270
271     @Override
272     public List<WritingGroup> listWritingGroups() throws SQLException {
273         Statement stmt = con.createStatement();
274         List<WritingGroup> groups = new LinkedList<>();
275         ResultSet results = stmt.executeQuery(SQL_GET_GROUPS);
276
277         while (results.next()) {
278             groups.add(new WritingGroup(results.getString(1), results.getString(2),
    results.getString(3),
279                     results.getString(4)));
280         }
```

```
281
282         results.close();
283
284         return groups;
285     }
286
287     @Override
288     public WritingGroup getWritingGroup(String name) throws SQLException {
289         this.PSTMT_GET_WRITING_GROUP.setString(1, name);
290         ResultSet result = this.PSTMT_GET_WRITING_GROUP.executeQuery();
291
292         if (result.next()) {
293             WritingGroup group = new WritingGroup(result.getString(1), result.getString(2),
    result.getString(3),
294                     result.getString(4));
295             result.close();
296             return group;
297         }
298
299         return null;
300     }
301
302     @Override
303     public void insertWritingGroup(WritingGroup group) throws
    SQLIntegrityConstraintViolationException, SQLException {
304         this.PSTMT_INSERT_WRITING_GROUP.setString(1, group.groupName);
305         this.PSTMT_INSERT_WRITING_GROUP.setString(2, group.headWriter);
306         this.PSTMT_INSERT_WRITING_GROUP.setString(3, group.yearFormed);
307         this.PSTMT_INSERT_WRITING_GROUP.setString(4, group.subject);
308         this.PSTMT_INSERT_WRITING_GROUP.executeUpdate();
309     }
310
311     @Override
312     public void deleteWritingGroup(String groupName) throws
    SQLIntegrityConstraintViolationException, SQLException {
313         this.PSTMT_DELETE_WRITING_GROUP.setString(1, groupName);
314         this.PSTMT_DELETE_WRITING_GROUP.executeUpdate();
315     }
316
317     /**
318      * Checks the given string is a valid year string, that is, exactly 4
319      * characters in length, all of which are digits.
320      *
321      * @param year the string to check
322      * @return is year valid
323      */
324     public static boolean checkYearString(String year) {
325         if (year.length() > 4) {
326             return false;
327         }
328         for (int i = 0; i < year.length(); i++) {
329             if (!Character.isDigit(year.charAt(i))) {
330                 return false;
331             }
332         }
333
334         return true;
```

```
335     }
336
337 }
338
```

```java

package cecs323.jdbcproject.pojos;

/**
 * The BookDetail class combines the {@link Book}, {@link Publisher}, and
 * {@link WritingGroup} classes into one object.
 *
 * @author Nicholas
 */
public class BookDetail {
    /**
     * The {@link Book} that this <code>BookDetails</code> stores details of.
     */
    public final Book book;

    /**
     * The {@link Publisher} of {@link #book}.
     */
    public final Publisher publisher;

    /**
     * The {@link WritingGroup} that wrote {@link #book}.
     */
    public final WritingGroup writingGroup;

    /**
     * Creates a new <code>BookDetail</code> for the given {@link Book} with the
     * given {@link Publisher} and {@link WritingGroup}.
     *
     * @param b the book to detail
     * @param p the publisher of the book
     * @param wg the writing group of the book
     */
    public BookDetail(Book b, Publisher p, WritingGroup wg) {
        this.book = b;
        this.publisher = p;
        this.writingGroup = wg;
    }
}

```

```java
package cecs323.jdbcproject.pojos;

/**
 * The Book class is a POJO (Plain Old Java Object) that is used to encapsulate
 * the attributes of an entry in the Books table.
 *
 * @author Nicholas
 */
public class Book {
    /**
     * The name of the {@link WritingGroup} that wrote this Book.
     *
     * Must be no more than 30 characters in length, and the name of an existing
     * <code>WritingGroup</code>.
     */
    public String groupName;

    /**
     * The title of this Book.
     *
     * Must be no more than 40 characters in length, and unique within the
     * {@link WritingGroup} that wrote this Book.
     *
     * @see #groupName
     */
    public String bookTitle;

    /**
     * The name of the {@link Publisher} that published this Book.
     *
     * Must be no more than 30 characters in length, ane equivalent to the name
     * of an existing <code>Publisher</code>.
     */
    public String publisherName;

    /**
     * The year in which this Book was published.
     *
     * Must be exactly 4 characters in length.
     */
    public String yearPublished;

    /**
     * The number of pages in this Book.
     */
    public int numberPages;

    /**
     * Creates a new Book object with the given values.
     *
     * @param title the title of the book
     * @param groupName the name of the WritingGroup that wrote this book
     * @param pubName the name of the publisher that published this book
     * @param year the year that this book was published
     * @param pages the number of pages in this book
     */
    public Book(String title, String groupName, String pubName, String year, int pages) {
```

```
58          this.bookTitle = title;
59          this.groupName = groupName;
60          this.publisherName = pubName;
61          this.numberPages = pages;
62          this.yearPublished = year;
63      }
64  }
65
```

```java
package cecs323.jdbcproject.pojos;

/**
 * The BookKeyData class is a POJO (Plain Old Java Object) that stores the
 * primary key attributes of an entry in the Books table.
 *
 * @author Nicholas
 */
public class BookKeyData {
    /**
     * The title of the book.
     */
    public String bookTitle;

    /**
     * The name of the writing group that wrote the book.
     */
    public String writingGroup;

    /**
     * Constructor for BookKeyData
     *
     * @param bookTitle title of a book
     * @param writingGroup name of a writing group
     */
    public BookKeyData(String bookTitle, String writingGroup) {
        this.bookTitle = bookTitle;
        this.writingGroup = writingGroup;
    }
}

```

```java
package cecs323.jdbcproject.pojos;

/**
 * The Publisher class is a POJO (Plain Old Java Object) used to encapsulate the
 * attributes of an entry in the Publishers table.
 *
 * @author Nicholas
 */
public class Publisher {
    /**
     * The name of this Publisher.
     *
     * Must be no more than 30 characters in length.
     */
    public String publisherName;

    /**
     * The address of this Publisher.
     *
     * Must be no more than 30 characters in length.
     */
    public String publisherAddress;

    /**
     * The phone number of this Publisher.
     *
     * Must be no more than 20 characters in length.
     */
    public String publisherPhone;

    /**
     * The email address of this Publisher.
     *
     * Must be no more than 50 characters in length.
     */
    public String publisherEmail;

    /**
     * Creates a new Publisher object with the given name, address, phone
     * number, and email address.
     *
     * @param name the name of the publisher
     * @param address the address of the publisher
     * @param phone the phone number of the publisher
     * @param email the email address of the publisher
     */
    public Publisher(String name, String address, String phone, String email) {
        this.publisherName = name;
        this.publisherAddress = address;
        this.publisherPhone = phone;
        this.publisherEmail = email;
    }
}

```

```java
package cecs323.jdbcproject.pojos;

/**
 * The WritingGroup class is a POJO (Plain Old Java Object) that is used to
 * compartmentalize the attributes of an entry in the WritingGroups table.
 *
 * @author Nicholas
 */
public class WritingGroup {
    /**
     * The name of the WritingGroup.
     *
     * Must be no more than 30 characters in length.
     */
    public String groupName;

    /**
     * The name of the head writer in this WritingGroup.
     *
     * Must be no more than 30 characters in length.
     */
    public String headWriter;

    /**
     * The year that this WritingGroup was formed.
     *
     * Must be exactly 4 characters in length.
     */
    public String yearFormed;

    /**
     * The subject that this WritingGroup writes about.
     *
     * Must be no more than 50 characters in length.
     */
    public String subject;

    /**
     * Creates a new WritingGroup object with the given name, head writer, year
     * and subject.
     *
     * @param name the name of the writing group
     * @param head the head writer of the writing group
     * @param year the year in which the group formed
     * @param subj the subject that the group writes about
     */
    public WritingGroup(String name, String head, String year, String subj) {
        this.groupName = name;
        this.headWriter = head;
        this.yearFormed = year;
        this.subject = subj;
    }
}

```

```sql
CREATE TABLE publishers(
    publishername VARCHAR(40) NOT NULL,
    publisheraddress VARCHAR(40),
    publisherphone VARCHAR(40),
    publisheremail VARCHAR(40),
    CONSTRAINT publisher_pk PRIMARY KEY (publishername)
);
CREATE TABLE writinggroups(
    groupname VARCHAR(40) NOT NULL,
    headwriter VARCHAR(40),
    yearformed VARCHAR(40),
    subject VARCHAR(40),
    CONSTRAINT writinggroup_pk PRIMARY KEY (groupname)
);
CREATE TABLE books(
    groupname VARCHAR(40) NOT NULL,
    booktitle VARCHAR(40) NOT NULL,
    publishername VARCHAR(40) NOT NULL,
    yearpublished VARCHAR(40),
    numberpages INT,
    CONSTRAINT books_pk PRIMARY KEY (groupname, booktitle),
    CONSTRAINT publishers_books_FK FOREIGN KEY (publishername) REFERENCES
    publishers(publishername),
    CONSTRAINT writinggroup_books_FK FOREIGN KEY (groupname) REFERENCES
    writinggroups(groupname)
);
--INSERT VALUE IN publishers--
INSERT INTO publishers (publishername, publisheraddress, publisherphone,
publisheremail) VALUES
('Apple Inc.','New York, New York','617501426','apple@live.com'),
('Orange Inc.','Cape Town, South Africa','901524395','orange@hotmail.com'),
('Mango Inc.','Tokyo, Japan','444370190','mango@gmail.com');

--INSERT VALUE INTO writinggroups--
INSERT INTO writinggroups (groupname, headwriter, yearformed, subject) VALUES
('Alpha','John','1538','English'),
('Omega','Lucy','1982','History'),
('Epsilon','Marry','2007','Chemistry'),
('Delta','Zake','2014','Physic'),
('Beta','Niko','1753','Math');


--INSERT VALUE INTO books--
INSERT INTO books (groupname, booktitle, publishername, yearpublished,
numberpages)VALUES
('Alpha','In Search of Lost Time by Marcel Proust','Orange Inc.','1563',156),
('Omega','Don Quixote by Miguel de Cervantes','Orange Inc.','1792',387),
('Delta','Ulysses by James Joyce','Orange Inc.','2017',4562),
('Beta','The Odyssey by Homer','Apple Inc.','2000',1234),
('Epsilon','War and Peace by Leo Tolstoy','Mango Inc.','1982',7453),
('Epsilon','Moby Dick by Herman Melville','Mango Inc.','1970',4125),
('Beta','The Divine Comedy by Dante Alighieri','Orange Inc.','1999',127),
('Omega','Hamlet by William Shakespeare','Mango Inc.','2003',852),
('Delta','Madame Bovary by Gustave Flaubert','Apple Inc.','2005',465),
('Alpha','The Iliad by Homer','Apple Inc.','1985',1357);
```