**Persistence Framework for Bookstore Application**

Following are the use cases for this application:

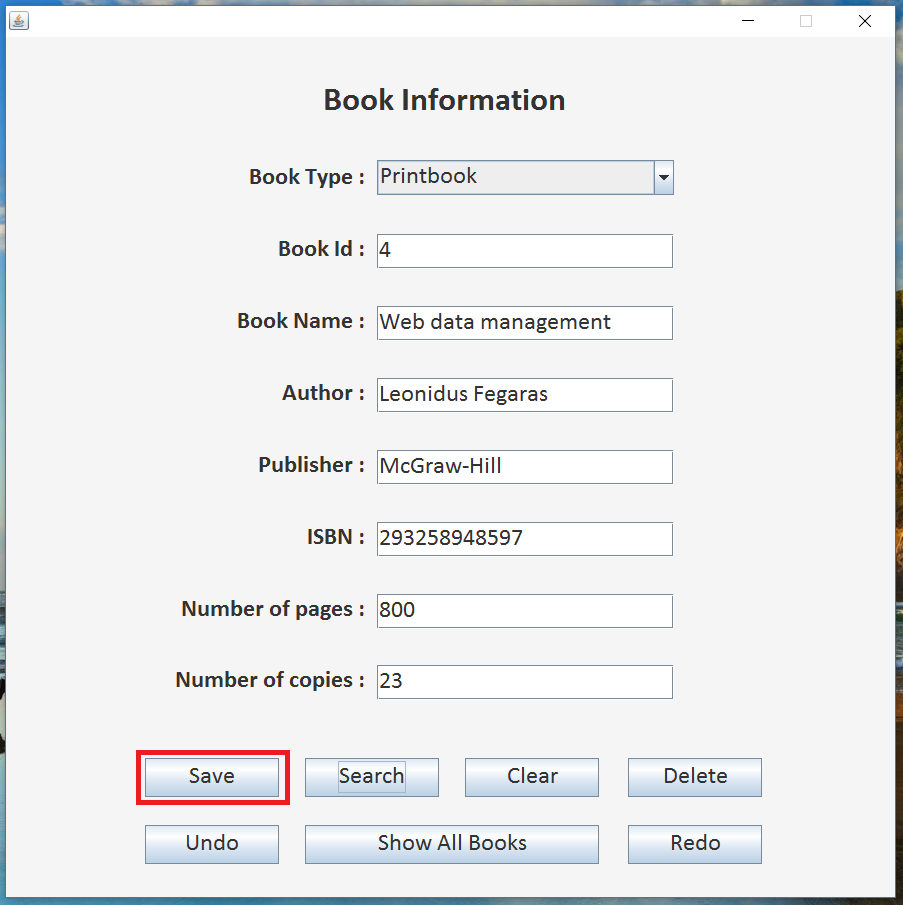
**Use cases:**

1. Save book
2. Search book
3. Delete book
4. Undo or redo the last operation

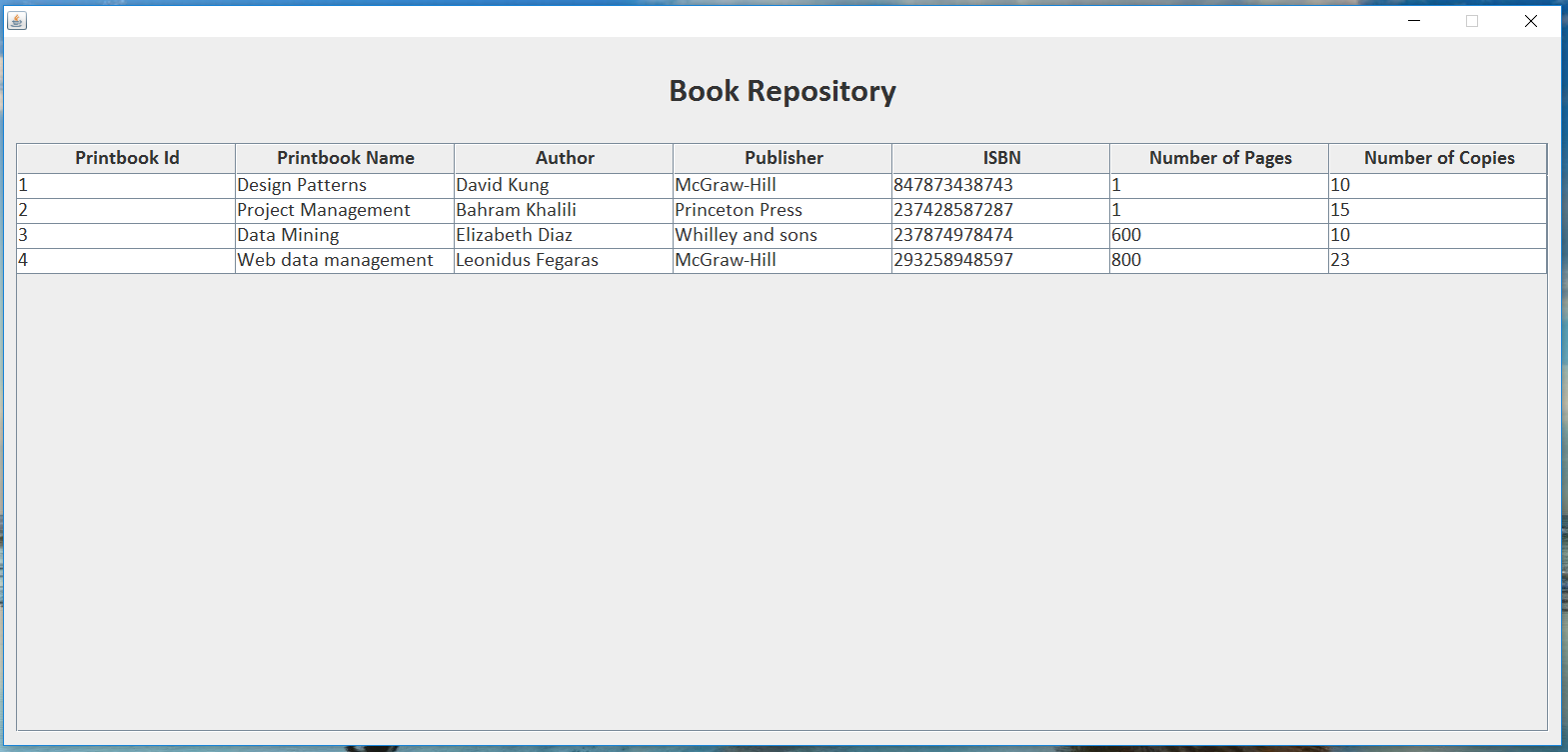
*(Note: Creating a new book is equivalent to saving a book. Also, updating a book information is equivalent to first searching, changing and then saving a book. Therefore, these are not treated as separate use cases.)*

1. **Save book:**
2. **How to save a book:**

Select the type of book (*Printbook* or *Ebook*) from the book type drop down. Then provide other details about the book. Finally, click ‘Save’ button.



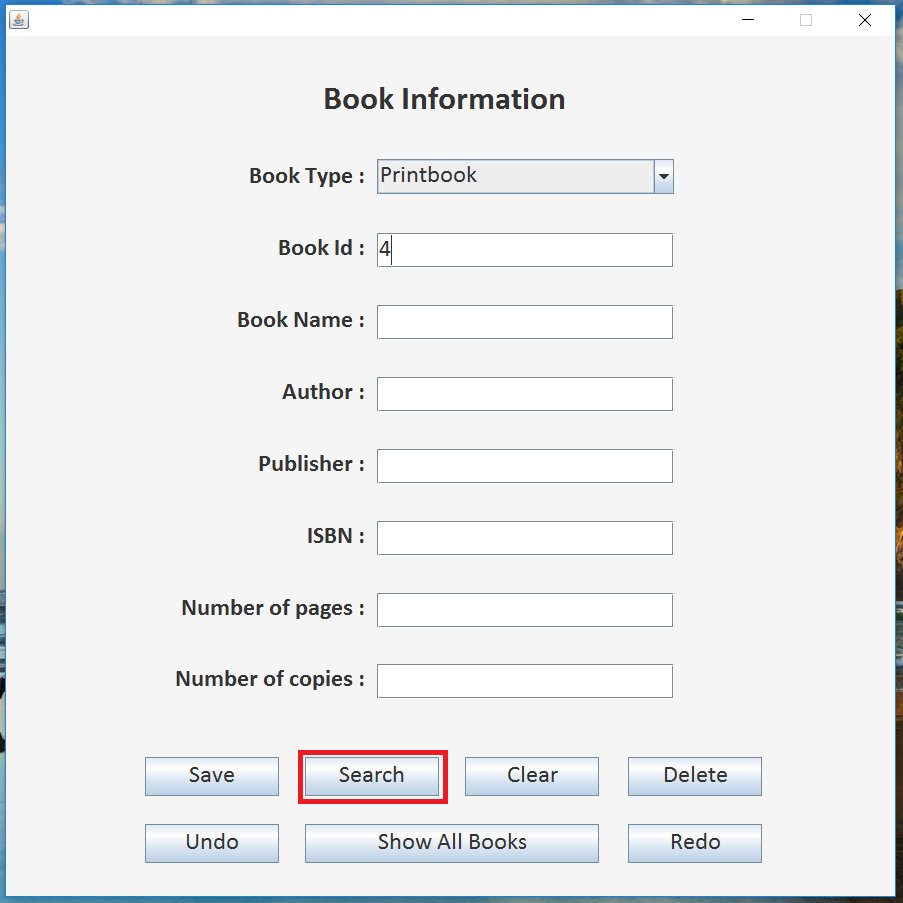
After this, the book information will be added in the database. User can verify it by clicking on ‘Show All Books’ button.



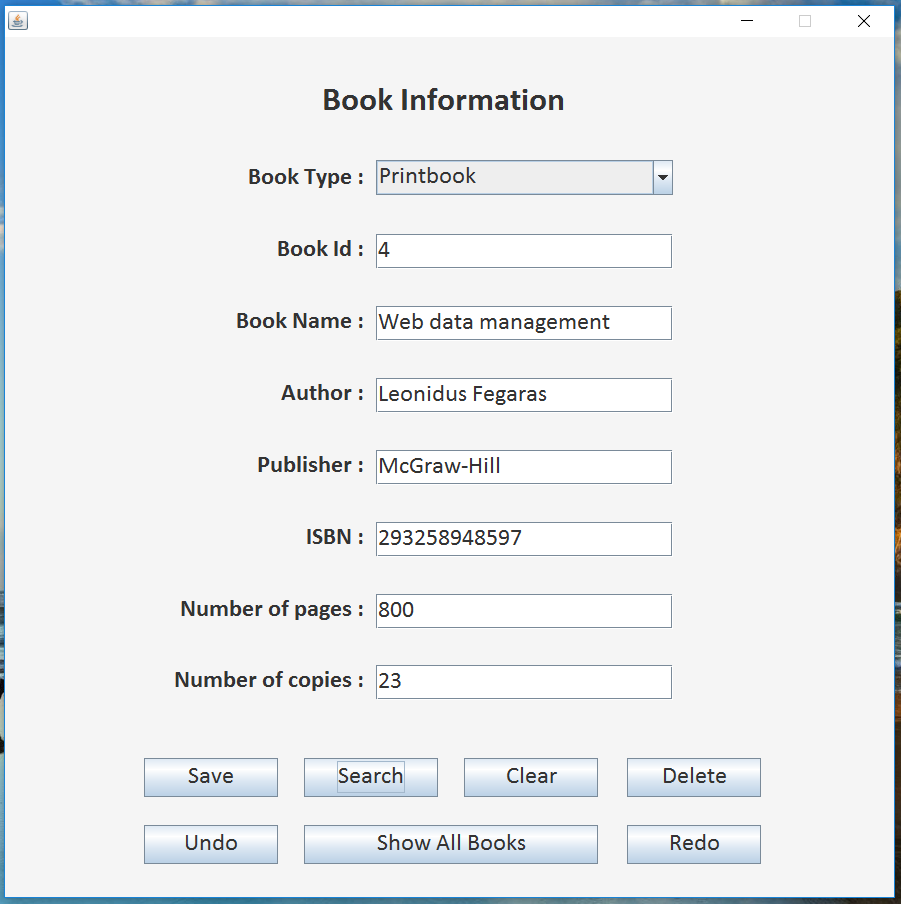
1. **How it works:**
2. Clicking on *Save* button invokes its action listener method.
3. The action listener calls the *saveBook* method of *SaveController* and passes the entered information to it.
4. *SaveContorller* creates a new instance of *DBManager* and invokes its *savePrintbook* or *saveEbook* method and passes the book object.
5. *DBManager* calls the *savePrintbook* or *saveEbook* method of *MysqlImpl*.
6. In *MysqlImpl*, a new command object for save (*SavePrintbook or SaveEbook*) is created and its execute method is called.
7. Finally, the command object is pushed on the execution stack.
8. **Search book:**
9. **How to search a book:**

Select the type of book (*Printbook* or *Ebook*) from the book type drop down. Then provide any information about book e.g. book name, author, etc. and click the *Search* button. You can enter part of the string, e.g. instead of typing Christopher you can type Chris. Also, the search is not case sensitive, so you can enter the information in upper, lower or mixed case.

In below screenshot, we have provided book id as 4 which will be the search criteria.



The output of search operation will return the first record from database matching this search criteria.

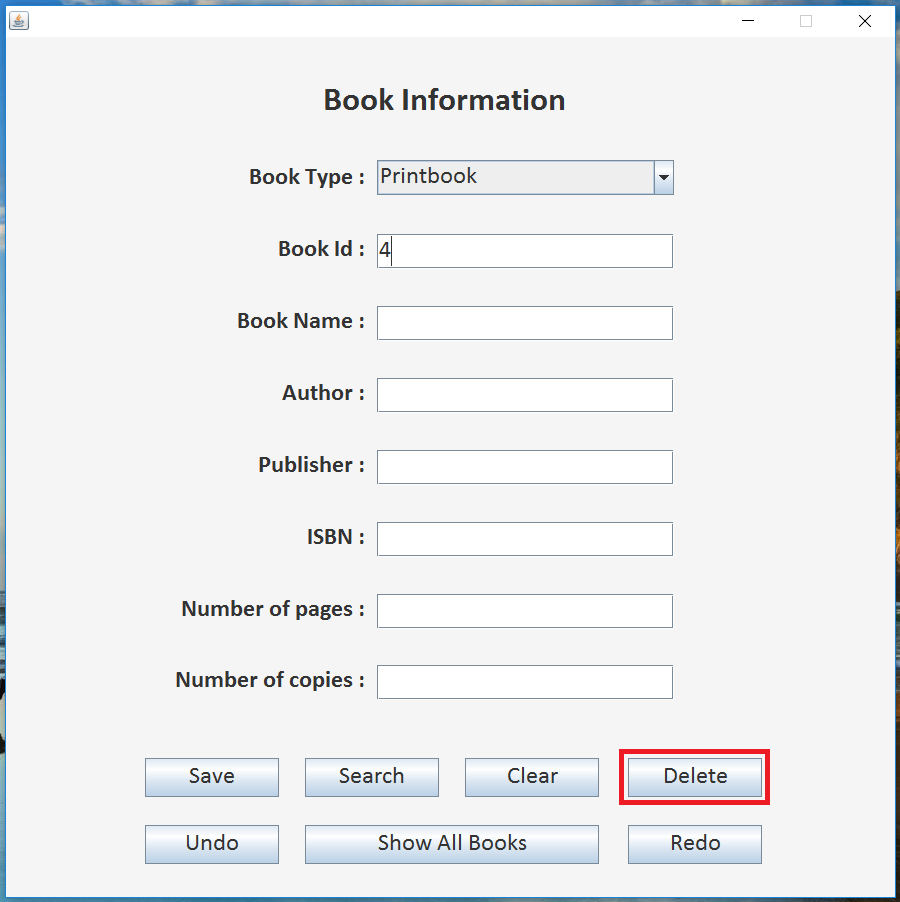


1. **How it works:**
2. Clicking on *Search* button invokes its action listener method.
3. The action listener calls the *searchBook* method of *SearchController* and passes the entered information to it.
4. *SearchController* creates a new instance of *DBManager* and invokes its *getPrintbook* or *getEbook* method and passes the *Printbook* or *Ebook* object.
5. *DBManager* calls the *getPrintbook* or *getEbook* method of *MysqlImpl*.
6. In *MysqlImpl*, a new command object for search (*GetPrintbook or GetEbook*) is created and its execute method is called.
7. The called method of *MysqlImpl will return an object of Printbook or Ebook to the DBManager.*
8. The *DBManager* will pass this object to the *SearchController*.
9. Finally, the *SearchController* will convert this object into string arraylist and sends it to user interface for display.

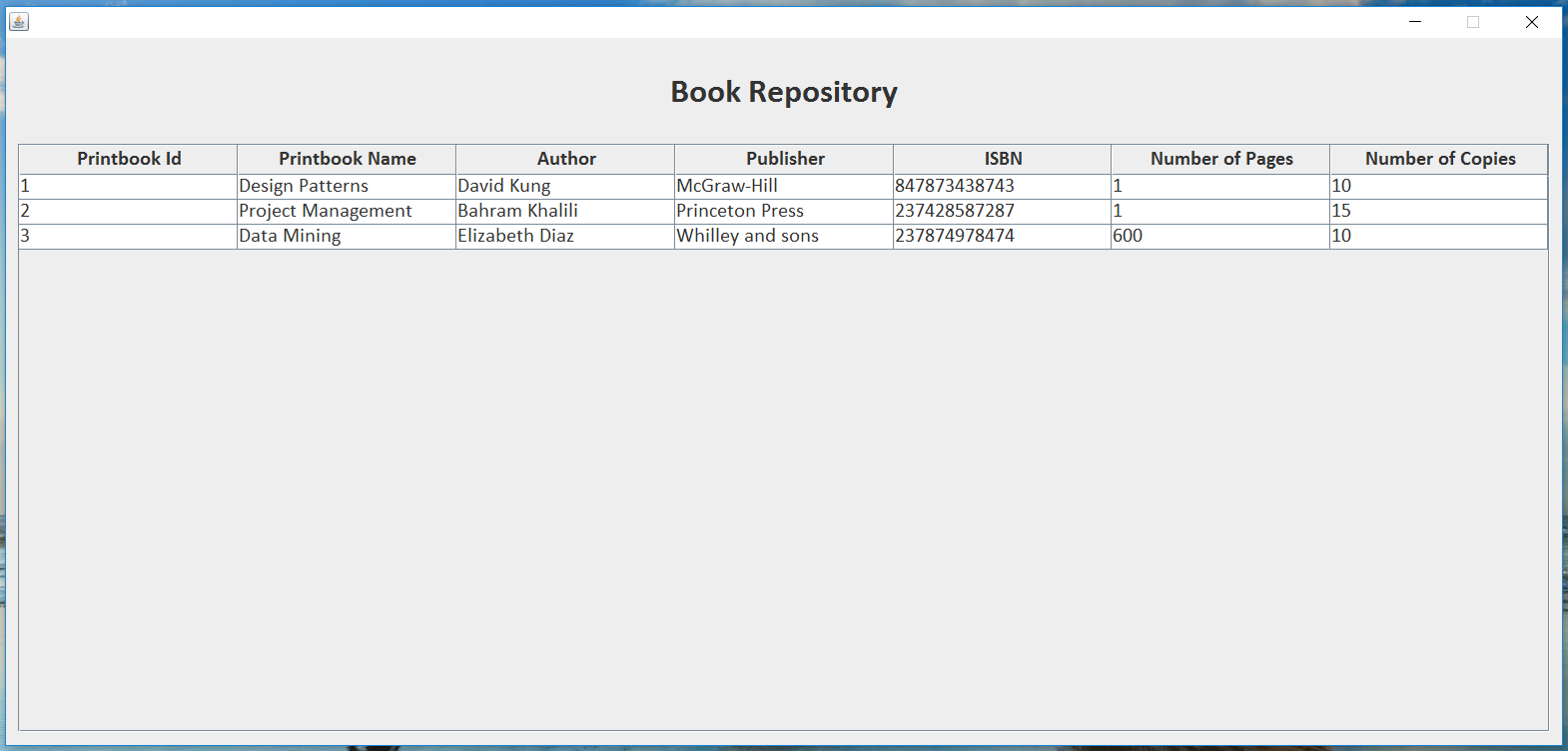
There is also a variant of this command which gets the information about all the books in the database. The working of this command is very similar as above except that some different functions and datatypes are used as we are dealing with a list of books and not a single book.

1. **Delete book:**
2. **How to delete a book:**

Select the type of book (*Printbook* or *Ebook*) from the book type drop down. Then enter the unique book id for the book and click the *Delete* button.

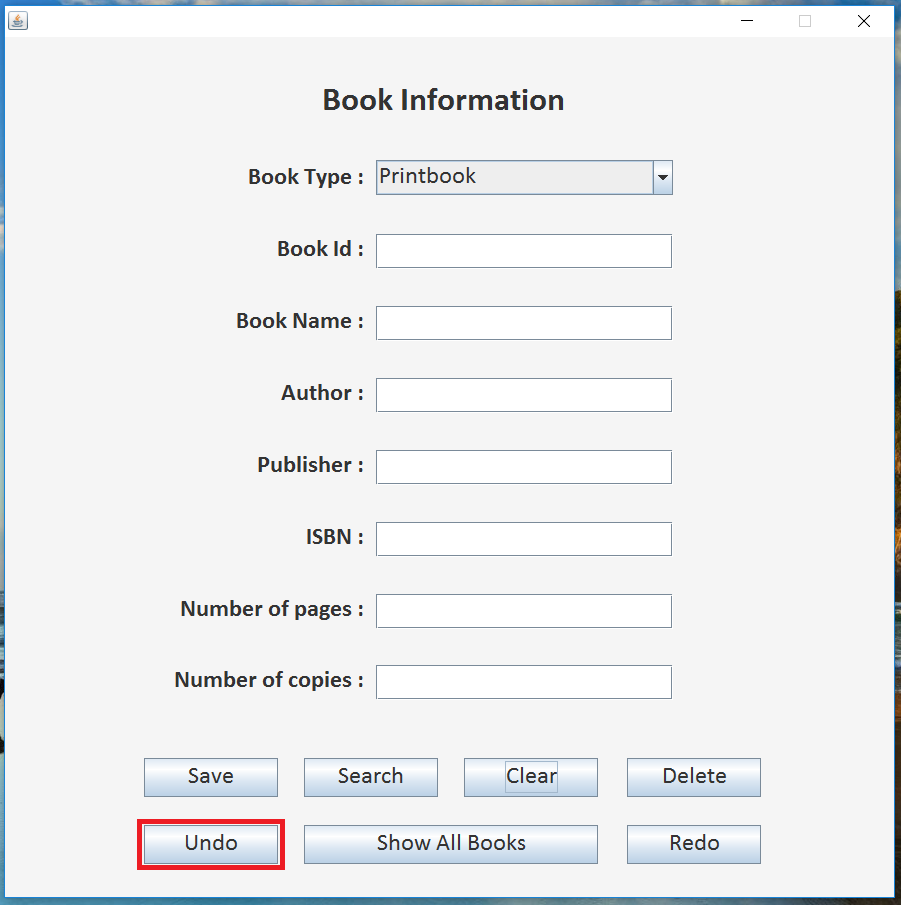


This will delete the information about the book having that id from database. User can verify it by clicking on ‘Show All Books’ button.

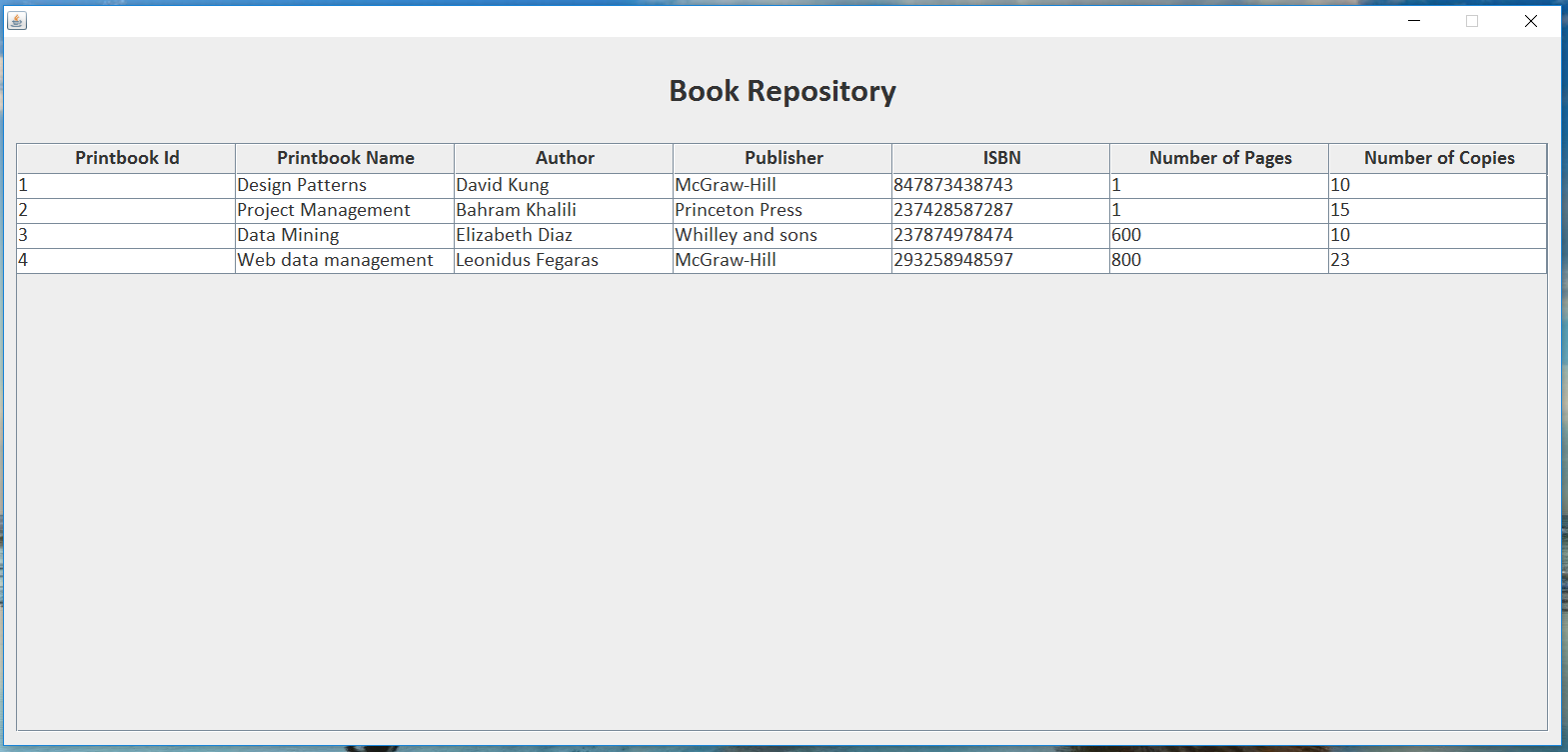


1. **How it works:**
2. Clicking on *Delete* button invokes its action listener method.
3. The action listener calls the *deleteBook* method of *DeleteController* and passes the book id to it.
4. *DeleteContorller* creates a new instance of *DBManager* and invokes its *deletePrintbook* or *deleteEbook* method and passes the book object.
5. *DBManager* calls the *deletePrintbook* or *deleteEbook* method of *MysqlImpl*.
6. In *MysqlImpl*, a new command object for delete (*DeleteEbook or DeletePrintbook*) is created and its execute method is called.
7. Finally, the command object is pushed on the execution stack.
8. **Undo or redo the last operation**
9. **How to undo an operation:**

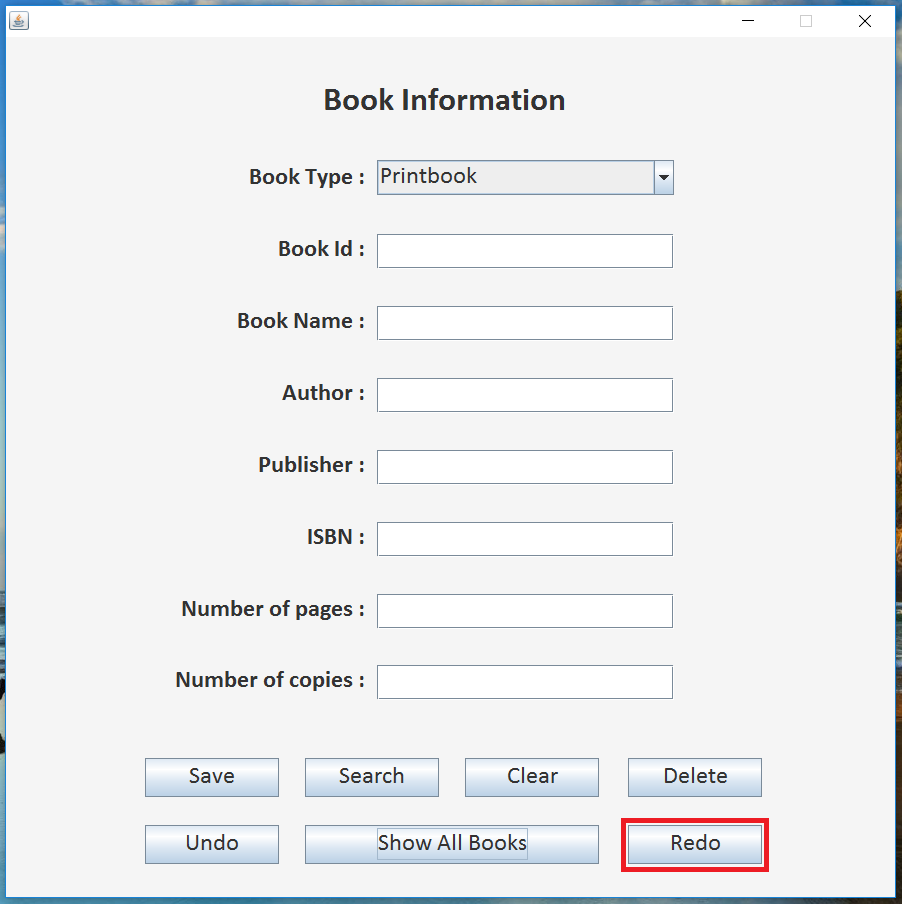
After performing a database update operation (e.g. we have deleted the book with book id 4), the user can click on the *Undo* buttonas shown on the user interface*.*



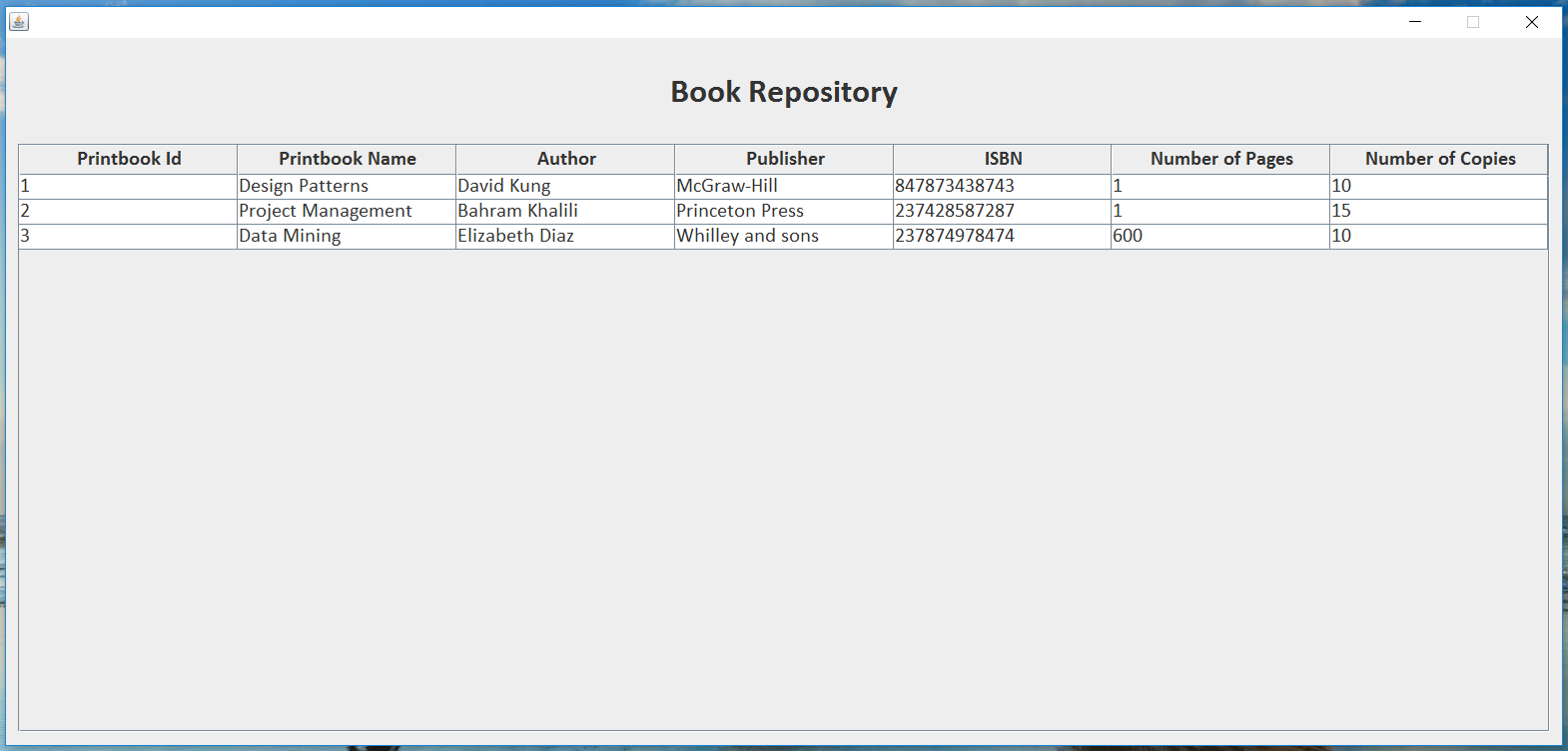
This will revert the effects of the last database update operation. So, in this case, the book will be inserted back into the database. User can verify this by clicking on ‘Show All Books’ button.



If user wants to re-apply the changes to the database, then he can click on the *Redo* button.



This will remove the effect of last undo command. In this case, the book will be deleted from the database again. User can verify this by clicking on ‘Show All Books’ button.



1. **How it works:**

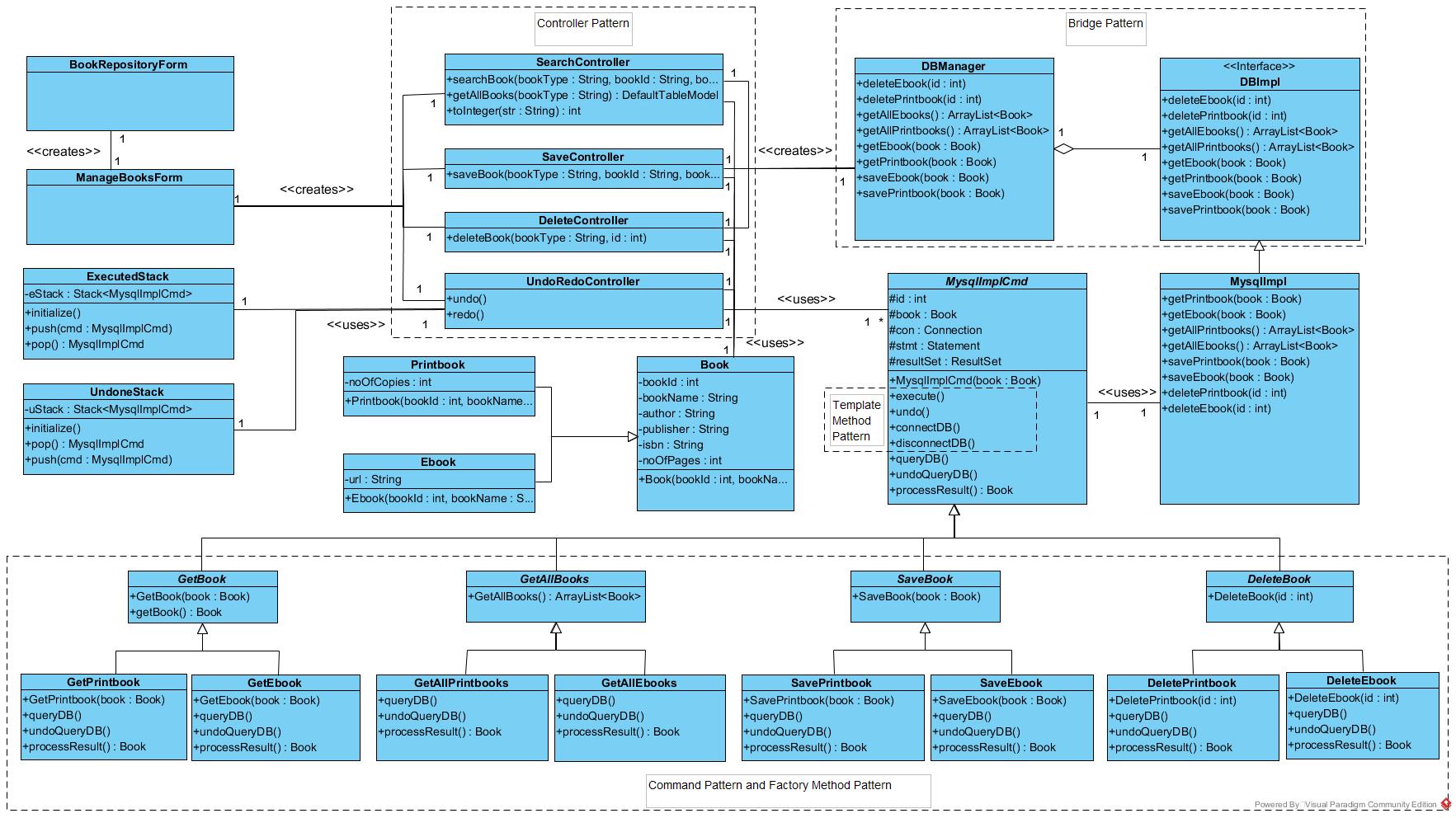
Undo:

1. Clicking on *Undo* button invokes its action listener method.
2. The action listener calls the *undo* method of *UndoRedoController.*
3. In this method, the controller pops a command from top of *ExecutedStack*. Then it calls the *undo* method of that command object. This will revert back the database updated made by that command object.
4. Then the command object is push on the top of *UndoneStack*.

Redo:

1. Clicking on *Redo* button invokes its action listener method.
2. The action listener calls the *redo* method of *UndoRedoController.*
3. In this method, the controller pops a command from top of *UndoneStack*. Then it calls the *execute* method of that command object. This will perform the database update operation of that command object.
4. Then the command object is push on the top of *ExecutedStack*.

**Design Class Diagram:**



**Patterns applied:**

Some problems were encountered while designing the project. In order to solve those projects, some design patterns were applied. Below is the information about design problems and patterns applied to solve those problems.

Design problem:

The system was required to be designed in such way that changing the underlying database will have minimum impact on the other parts of system.

Solution:

**Bridge** pattern was used to have database implementation loosely coupled with the other part of application. This way one can easily change the underlying database without having a large impact on its other parts.

Design problem:

The user interface was tightly coupled with the business objects and business access layer. Therefore, changing something in one part required changes in other parts as well.

Solution:

**Controller** pattern was applied to solve the problem. Four controllers namely SaveController, SearchController, DeleteController and UndoRedoController were created corresponding to the four use cases. These controller will receive user requests from user interface and pass those to the corresponding business object.

Design problem:

The system was required to have undo and redo functionality. This required flexibility in the execution sequence of various database operations.

Solution:

**Command** pattern was applied to encapsulate various database operations as command objects. This helped in providing the required flexibility in execution sequence.

Design problem:

While performing various database operations, some of the steps were common for all operations, but some steps were varying depending on the type of database operation.

Solution:

**Template methods** and hook methods were created in the MysqlImplCmd abstract class. The subclasses of MysqlImplCmd class provided the implementation for hook methods. This way we defined the common steps in superclass and steps specific to every database operation in subclasses.

Design problem:

There were two types books in the project as Printbook and Ebook. All the required functionality was same for both the objects. So, in this case, the algorithm was same but the object used by the algorithm were varying.

Solution:

**Factory method named** processResult was defined in the MysqlImplCmd class. This method returns object of book from different subclasses i.e. ebook and printbook.