```
###############
# Return.java #
###############

package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.models.SystemDateTime;
import lbms.models.Transaction;
import lbms.models.Visitor;
import lbms.search.UserSearch;

import java.util.ArrayList;
import java.util.List;

/**
 * Returns a book borrowed by a library visitor.
 * @author Team B
 */
public class Return implements Undoable {

    private long visitorID;
    private long clientID;
    private List<Integer> ids = new ArrayList<>();

    /**
     * Constructor for a Return command object.
     * @param request: the request input string
     */
    public Return(String request) {
        int count = 0;
        String[] arguments = request.split(",");
        for (int i = 0; !arguments[i].startsWith("{"); i++) {
            count++;
        }
        if (count == 1) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID =
LBMS.getSessions().get(this.clientID).getV().getVisitorID();
            for (int i = 1; i < arguments.length; i++) {
                if (arguments[i].startsWith("{")) {

this.ids.add(Character.getNumericValue(arguments[i].charAt(1)));
                } else if (arguments[i].endsWith("}")) {

this.ids.add(Character.getNumericValue(arguments[i].charAt(0)));
                } else {
                    this.ids.add(Integer.parseInt(arguments[i]));
                }
            }
        } else if (count == 2) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID = Long.parseLong(arguments[1]);
            for (int i = 2; i < arguments.length; i++) {
                if (arguments[i].startsWith("{")) {

this.ids.add(Character.getNumericValue(arguments[i].charAt(1)));
                } else if (arguments[i].endsWith("}")) {
```

```
this.ids.add(Character.getNumericValue(arguments[i].charAt(0)));
                } else {
                    this.ids.add(Integer.parseInt(arguments[i]));
                }
            }
        }
    }

    /**
     * Executes the return command.
     * @return a response string or error message
     */
    @Override
    public String execute() {
        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-visitor-id;";
        }
        Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
        ArrayList<Integer> nonBooks = new ArrayList<>();
        for (Integer id : this.ids) {
            if
(LBMS.getSessions().get(this.clientID).getBookSearch().size() >= id) {
                try {
                    Book b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
                    visitor.getCheckedOutBooks().get(b.getIsbn());
                } catch (Exception e) {
                    nonBooks.add(id);
                }
            } else {
                nonBooks.add(id);
            }
        }
        if (nonBooks.size() > 0) {
            String output = ",invalid-book-id,";
            LBMS.getSessions().get(clientID).popUndoable();
            for (Integer i: nonBooks) {
                output += i + ",";
            }
            output = output.replaceAll(",$", "");
            return output + ";";
        }

        if (visitor.getFines() > 0.0) {
            String output = ",overdue," + String.format("%.2f",
visitor.getFines()) + ",";
            for (Transaction t: visitor.getCheckedOutBooks().values()) {
                if
(SystemDateTime.getInstance(null).getDate().isAfter(t.getDueDate())) {
                    output +=
LBMS.getSessions().get(this.clientID).getBookSearch().indexOf(LBMS.getBoo
ks()
                            .get(t.getIsbn())) + 1 + ",";
                }
            }
            LBMS.getSessions().get(clientID).popUndoable();
            return output.replaceAll(",$", ";");
        }
```

```
        for (Integer i: this.ids) {
            Book b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(i - 1);
            b.returnBook();
            Transaction t =
visitor.getCheckedOutBooks().get(b.getIsbn());
            LBMS.getVisitors().get(visitorID).returnBook(t);
            t.closeTransaction();
        }

        return ",success;";
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it failed
     */
    @Override
    public String unExecute() {
        Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
        for (Integer id : this.ids) {
            Book b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
            b.undoReturnBook();

LBMS.getVisitors().get(this.visitorID).undoReturnBook(visitor.getPrevious
CheckedOutBooks().get(b.getIsbn()));
        }
        return null;
    }
}

####################
# GetDateTime.java #
####################

package lbms.command;

import lbms.models.SystemDateTime;

/**
 * GetDateTime class that calls the API to get the system time.
 * @author Team B
 */
public class GetDateTime implements Command {

    /**
     * Constructor for GetDateTime.
     */
    public GetDateTime() {}

    /**
     * Gets the system date and time.
     */
    @Override
    public String execute() {
        return "," +
SystemDateTime.getInstance(null).getDate().format(SystemDateTime.DATE_FOR
MAT) + "," +
```

```
SystemDateTime.getInstance(null).getTime().format(SystemDateTime.TIME_FOR
MAT) + ";";
    }
}

################
# Command.java #
################

package lbms.command;

/**
 * Interface for the Command design pattern.
 * @author Team B
 */
public interface Command {

    /**
     * Executes the command.
     * @return any parameter errors or null for success
     */
    String execute();

}

################
# PayFine.java #
################

package lbms.command;

import lbms.LBMS;
import lbms.search.UserSearch;

import java.text.DecimalFormat;

/**
 * PayFine class for the pay fine command.
 * @author Team B
 */
public class PayFine implements Undoable {

    private long clientID;
    private long visitorID;
    private double amount;

    /**
     * Constructor for a PayFine command object.
     * @param request: the request string to be processed
     */
    public PayFine(String request) {
        String[] arguments = request.split(",");
        if (arguments.length == 2) {
            this.clientID = Long.parseLong(arguments[0]);
            this.amount = Double.parseDouble(arguments[1]);
            this.visitorID =
LBMS.getSessions().get(this.clientID).getV().getVisitorID();
        } else if (arguments.length == 3) {
            this.clientID = Long.parseLong(arguments[0]);
```

```
            this.amount = Double.parseDouble(arguments[1]);
            this.visitorID = Long.parseLong(arguments[2]);
        }
    }

    /**
     * Executes the command for pay fine.
     * @return a response or error message
     */
    @Override
    public String execute() {
        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-visitor-id;";
        }
        double balance =
UserSearch.BY_ID.findFirst(visitorID).getFines();
        if (this.amount < 0 || this.amount > balance) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-amount," + this.amount + "," + new
DecimalFormat("#.00").format(balance) + ";";
        } else {
            double newBalance = balance - this.amount;

UserSearch.BY_ID.findFirst(this.visitorID).payFines(this.amount);
            return ",success," + new
DecimalFormat("#.00").format(newBalance) + ";";
        }
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it fails
     */
    @Override
    public String unExecute() {
        UserSearch.BY_ID.findFirst(this.visitorID).payFines(-
this.amount);
        return null;
    }
}

##################
# ResetTime.java #
##################

package lbms.command;

import lbms.models.SystemDateTime;

/**
 * ResetTime class used to reset the time during testing.
 * @author Team B
 */
public class ResetTime implements Command {

    /**
     * Constructor for ResetTime command.
     */
    public ResetTime() {}
```

```
    /**
     * Executes the reset time command on the system.
     * @return a string of the response
     */
    @Override
    public String execute() {
        try {
            SystemDateTime.getInstance(null).reset();
            return "success;";
        } catch (Exception e) {
            return "failure;";
        }
    }
}

###############
# LogOut.java #
###############

package lbms.command;

import lbms.LBMS;
import lbms.models.Session;

/**
 * LogOut class for the log out command.
 * @author Team
 */
public class LogOut implements Command {

    private Long clientID;

    /**
     * Constructor for a log out object.
     * @param clientID: the ID of the client to be logged out
     */
    public LogOut(Long clientID) {
        this.clientID = clientID;
    }

    /**
     * Executes the command by interacting with the backend in the LBMS.
     * @return a response as a string
     */
    public String execute() {
        Session session = LBMS.getSessions().get(this.clientID);
        if (session == null) {
            return ",invalid-client-id;";
        }
        session.setV(null);
        return ",success;";
    }
}

####################
# FindBorrowed.java #
####################
```

```
package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.models.Transaction;
import lbms.models.Visitor;
import lbms.search.BookSearch;
import lbms.search.UserSearch;

/**
 * Queries for a list of books currently borrowed by a specific visitor.
 * @author Team B
 */
public class FindBorrowed implements Command {

    private long visitorID;
    private long clientID;

    /**
     * Constructor for FindBorrowed class.
     * @param request: the request String for the command
     */
    public FindBorrowed(String request) {
        String[] arguments = request.split(",");
        if (arguments.length == 1) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID =
LBMS.getSessions().get(clientID).getV().getVisitorID();
        } else if (arguments.length == 2) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID = Long.parseLong(arguments[1]);
        }
    }

    /**
     * Executes the find borrowed command.
     * @return a response or error message
     */
    @Override
    public String execute() {
        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            return ",invalid-visitor-id;";
        }

        Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
        String s = "";
        s += visitor.getNumCheckedOut();
        final int[] id = {1};

        Book b;
        LBMS.getSessions().get(this.clientID).getBookSearch().clear();
        for (Transaction t: visitor.getCheckedOutBooks().values()) {
            b =
BookSearch.BY_ISBN.findAll(t.getIsbn().toString()).get(0);
            LBMS.getSessions().get(this.clientID).getBookSearch().add(b);
            s += ",\n" + id[0]++ + "," + t.getIsbn() + "," + b.getTitle()
+ "," + t.getDate();
        }
        return "," + s + ";";
    }
```

```
}

######################
# CreateAccount.java #
######################

package lbms.command;

import lbms.LBMS;
import lbms.models.Employee;
import lbms.models.Visitor;

/**
 * CreateAccount class for the create account command.
 * @author Team B
 */
public class CreateAccount implements Command {

    private String username;
    private String password;
    private String role;
    private Long visitorID;

    /**
     * Constructor for the CreateAccount command.
     * @param request: the string for input
     * @throws MissingParametersException: when the request format is
invalid
     */
    public CreateAccount(String request) throws
MissingParametersException {
        String[] arguments = request.split(",");
        if (arguments.length == 1 && arguments[0].equals("")) {
            throw new MissingParametersException("missing-
parameters,{all};");
        } else if (arguments.length == 1) {
            throw new MissingParametersException("missing-
parameters,{password,role,visitorID};");
        } else if (arguments.length == 2) {
            throw new MissingParametersException("missing-
parameters,{role,visitorID};");
        } else if (arguments.length == 3) {
            throw new MissingParametersException("missing-
parameters,{visitorID};");
        } else {
            this.username = arguments[0];
            this.password = arguments[1];
            this.role = arguments[2];
            try {
                this.visitorID = Long.parseLong(arguments[3]);
            } catch (NumberFormatException e) {
                throw new MissingParametersException("invalid-visitor;");
            }
        }
    }

    /**
     * Processes the models of the LBMS based on the command.
     * @return the response string
     */
```

```
    @Override
    public String execute() {
        // perform error checks
        if (usernameExists()) {
            return ",duplicate-username;";
        }

        Visitor v = LBMS.getVisitors().get(this.visitorID);
        if (v == null) {
            return ",invalid-visitor;";
        }
        if (accountExists(v)) {
            LBMS.getVisitors().put(v.getVisitorID(), v);
            return ",duplicate-visitor;";
        }
        // add the visitor/employee to LBMS
        if (this.role.toLowerCase().equals("visitor") ||
this.role.toLowerCase().equals("employee")) {
            LBMS.getVisitors().put(this.visitorID, v);
            if (this.role.toLowerCase().equals("employee")) {
                LBMS.getEmployees().put(this.visitorID, new Employee(v));
            }
        } else {
            return ",invalid-role;";
        }
        v.setCredentials(this.username, this.password);
        return ",success;";
    }

    /**
     * Checks if the username already exists in the system
     * @return true if username exists, false otherwise
     */
    private boolean usernameExists() {
        for (Visitor v: LBMS.getVisitors().values()) {
            if (v.getUsername() != null &&
v.getUsername().equals(this.username)) {
                return true;
            }
        }
        return false;
    }

    /**
     * Checks if the visitor already has an account
     * @param v visitor trying to make an account
     * @return true if account already exists, false otherwise
     */
    private boolean accountExists(Visitor v) {
        return v.getUsername() != null && v.getPassword() != null;
    }
}

#################
# EndVisit.java #
#################

package lbms.command;

import lbms.LBMS;
```

```
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.models.SystemDateTime;
import lbms.models.Visit;
import lbms.models.Visitor;
import lbms.search.UserSearch;

/**
 * EndVisit class for end visit command.
 * @author Team B
 */
public class EndVisit implements Undoable {

    private long clientID;
    private long visitorID;
    private Visit visit;

    /**
     * Constructor for an EndVisit command class.
     * @param request: request string holding clientID and visitorID
     */
    public EndVisit(String request) {
        String[] arguments = request.split(",");
        if (arguments.length == 1) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID =
LBMS.getSessions().get(this.clientID).getV().getVisitorID();
        } else if (arguments.length == 2) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID = Long.parseLong(arguments[1]);
        }
        this.visit = null;
    }

    /**
     * Executes the EndVisit command.
     * @return the response string or error message
     */
    @Override
    public String execute() {
        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-id;";
        }

        if (!ProxyCommandController.assistanceAuthorized(this.visitorID,
this.clientID)) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",not-authorized;";
        }

        if (UserSearch.BY_ID.findFirst(this.visitorID) != null) {
            Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
            if (visitor != null ) {
                if (visitor.getInLibrary()) {
                    this.visit =
LBMS.getCurrentVisits().remove(visitor.getVisitorID());
                    this.visit.depart();
                    LBMS.getTotalVisits().add(this.visit);
                    long s = this.visit.getDuration().getSeconds();
```

```
                    String duration = String.format("%02d:%02d:%02d", s /
3600, (s % 3600) / 60, (s % 60));
                    return "," + String.format("%010d", this.visitorID) +
"," +

this.visit.getDepartureTime().format(SystemDateTime.TIME_FORMAT) + "," +
duration + ";";
                }
                LBMS.getSessions().get(clientID).popUndoable();
                return ",invalid-id;";
            }
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-id;";
        }
        LBMS.getSessions().get(clientID).popUndoable();
        return ",invalid-id;";
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it fails
     */
    @Override
    public String unExecute() {
        this.visit.unDepart();
        LBMS.getCurrentVisits().put(this.visitorID, this.visit);
        LBMS.getTotalVisits().remove(this.visit);
        return null;
    }
}
```

```
################
# Invalid.java #
################

package lbms.command;

/**
 * Invalid command class.
 * @author Team B
 */
public class Invalid implements Command {

    /**
     * Constructor for an Invalid command.
     */
    public Invalid() {}

    /**
     * Executes the command.
     * @return string of the response
     */
    public String execute() {
        return "invalid-command;";
    }
}
```

```
######################
# ClientConnect.java #
######################
```

```
package lbms.command;

import lbms.LBMS;
import lbms.models.Session;

/**
 * ClientConnect class for the client connect command.
 * @author Team B
 */
public class ClientConnect implements Command {

    private Session s;

    /**
     * Constructor parses the request string and creates the necessary
data in the class.
     */
    public ClientConnect() {
        this.s = new Session();
    }

    /**
     * Executes the command by altering the models in the LBMS as
necessary.
     * @return the response for the system
     */
    @Override
    public String execute() {
        LBMS.getSessions().put(this.s.getClientID(), this.s);
        return s.getClientID() + "";
    }
}

##################
# Undoable.java #
##################

package lbms.command;

/**
 * Undoable interface for the commands that can be undone.
 * @author Team B
 */
public interface Undoable extends Command {

    /**
     * Un-executes the command, reverses the execute for that given
command.
     * @return a string if failure, null if success
     */
    String unExecute();

}

####################
# StoreSearch.java #
####################

package lbms.command;
```

```
import lbms.LBMS;
import lbms.models.Book;
import lbms.models.ISBN;
import lbms.search.BookSearch;
import lbms.search.GoogleAPISearch;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import static lbms.LBMS.SearchService.GOOGLE;
import static lbms.LBMS.SearchService.LOCAL;

/**
 * StoreSearch class that implements the book store search command.
 * @author Team B
 */
public class StoreSearch implements Command {

    private long clientID;
    private String title;
    private ArrayList<String> authors;
    private ISBN isbn;
    private String publisher = null;
    private String sortOrder = null;

    /**
     * Constructor for a StoreSearch object.
     * @param clientID: the clientID
     * @param request: the request string to be parsed
     * @throws MissingParametersException: when the request format is
invalid
     */
    public StoreSearch(long clientID, String request) throws
MissingParametersException {
        this.clientID = clientID;
        String[] arguments = request.split(",");
        if (arguments.length == 1 && arguments[0].equals("")) {
            throw new MissingParametersException("missing-
parameters,title;");
        }
        try {
            for (int index = 0; index < arguments.length; index++) {
                if (this.sortOrder == null &&
(Arrays.asList(arguments).contains("title") ||
                        Arrays.asList(arguments).contains("publish-
date"))) {
                    this.sortOrder = arguments[arguments.length - 1];
                }

                if (arguments[index].startsWith("{")) {
                    this.authors = new ArrayList<>();
                    while (!arguments[index].endsWith("}")) {

this.authors.add(arguments[index++].replaceAll("[{}]", ""));
                    }
                    this.authors.add(arguments[index].replaceAll("[{}]",
""));
                } else if (!arguments[index].equals("*")) {
```

```
                    if (this.title == null && !arguments[0].equals("*"))
{
                        this.title = (arguments[index]);
                    } else if (this.isbn == null &&
arguments[index].matches("^\\d{13}$")) {
                        this.isbn = new ISBN(arguments[index]);
                    } else if ((this.publisher == null && this.sortOrder
== null && index == (arguments.length) - 1) ||
                            (this.publisher == null && this.sortOrder !=
null && index == (arguments.length) - 2)) {
                        this.publisher = arguments[index];
                    }
                }
            }
        } catch (Exception e) {
            throw new MissingParametersException("unknown-error");
        }
    }

    /**
     * Executes the command for book store search.
     * @return a response or error string
     */
    @Override
    public String execute() {
        if (this.sortOrder != null && !this.sortOrder.equals("title") &&
!this.sortOrder.equals("publish-date")) {
            return "invalid-sort-order";
        }

        List<Book> books;
        if (LBMS.getSessions().get(this.clientID).getSearch() == LOCAL) {
            if (this.title != null) {
                books = BookSearch.BY_TITLE.toBuy().findAll(this.title);
            } else if (this.authors != null) {
                books =
BookSearch.BY_AUTHOR.toBuy().findAll(this.authors.get(0));
            } else if (this.isbn != null) {
                books =
BookSearch.BY_ISBN.toBuy().findAll(isbn.toString());
            } else if (this.publisher != null) {
                books =
BookSearch.BY_PUBLISHER.toBuy().findAll(this.publisher);
            } else {
                books = new ArrayList<>();
            }
        } else if (LBMS.getSessions().get(clientID).getSearch() ==
GOOGLE) {
            if (this.title != null) {
                books = GoogleAPISearch.searchByTitle(this.title);
            } else if (this.authors != null) {
                books = GoogleAPISearch.searchByAuthor(this.authors);
            } else if (this.isbn != null) {
                books =
GoogleAPISearch.searchByISBN(this.isbn.toString());
            } else if (this.publisher != null) {
                books =
GoogleAPISearch.searchByPublisher(this.publisher);
            } else {
                books = new ArrayList<>();
```

```
            }
        } else {
            books = new ArrayList<>();
        }

        List<Book> remove = new ArrayList<>();
        if (this.authors != null) {
            for (Book b: books) {
                for (String author: this.authors) {
                    if (!b.hasAuthorPartial(author)) {
                        remove.add(b);
                    }
                }
            }
        }
        if (this.isbn != null) {
            for (Book b: books) {
                if (!b.getIsbn().equals(this.isbn)) {
                    remove.add(b);
                }
            }
        }
        if (this.publisher != null) {
            for (Book b: books) {
                if (!b.getPublisher().contains(this.publisher)) {
                    remove.add(b);
                }
            }
        }
        books.removeAll(remove);

        if (this.sortOrder != null && this.sortOrder.equals("title")) {
            books.sort((Book b1, Book b2) ->
b2.getTitle().compareTo(b1.getTitle()));
        } else if (this.sortOrder != null &&
this.sortOrder.equals("publish-date")) {
            books.sort((Book b1, Book b2) ->
b2.getPublishDate().compareTo(b1.getPublishDate()));
        }

        if (books.size() == 0) {
            return ",0;";
        } else {
            int id = 1;
            StringBuilder response = new
StringBuilder(Integer.toString(books.size()) + ",\n");

LBMS.getSessions().get(this.clientID).getBookSearch().clear();
            for (Book book: books) {

LBMS.getSessions().get(this.clientID).getBookSearch().add(book);
                response.append(id).append(",")
                        .append(book.getIsbn()).append(",")
                        .append(book.getTitle()).append(",{");
                for (String author: book.getAuthors()) {
                    response.append(author).append(",");
                }
                response = new
StringBuilder(response.toString().replaceAll(",$", "},"));
                response.append(book.dateFormat()).append(",");
```

```
                response.append(book.getPageCount()).append(",\n");
                id += 1;
            }
            response = new StringBuilder(response.substring(0,
response.length() - 2));
            response.append(";");
            return "," + response.toString();
        }
    }
}

##############
# LogIn.java #
##############

package lbms.command;

import lbms.LBMS;
import lbms.models.Visitor;

/**
 * LogIn class for login command.
 * @author Team B
 */
public class LogIn implements Command {

    private Long clientID;
    private String username;
    private String password;

    /**
     * Constructor for a LogIn class object.
     * @param request: the request string to be processed
     * @throws MissingParametersException: when the request format is
invalid
     */
    public LogIn(String request) throws MissingParametersException {
        String parts[] = request.split(",");
        if (parts.length == 1) {
            throw new MissingParametersException("missing-
parameters,{all};");
        } else if (parts.length == 2) {
            throw new MissingParametersException("missing-
parameters,{password};");
        }
        this.clientID = Long.parseLong(parts[0]);
        this.username = parts[1];
        this.password = parts[2];
    }

    /**
     * Executes the command using the information from the request.
     * @return a string of the response to the system
     */
    @Override
    public String execute() {
        for (Visitor v: LBMS.getVisitors().values()) {
            if (v.getUsername() != null &&
v.getUsername().equals(this.username) &&
                    v.getPassword().equals(this.password)) {
```

```
                LBMS.getSessions().get(this.clientID).setV(v);
                return ",success;";
            }
        }
        return ",bad-username-or-password;";
    }
}

#############
# Redo.java #
#############

package lbms.command;

import lbms.LBMS;

/**
 * Redo class for the redo command.
 * @author Team B
 */
public class Redo implements Command {

    private Long clientID;

    /**
     * Constructor for a Redo class object.
     * @param clientID: the ID of the client
     */
    public Redo(Long clientID) {
        this.clientID = clientID;
    }

    /**
     * Processes the command by interacting with the backend in the LBMS.
     * @return a string of the response to the system
     */
    @Override
    public String execute() {
        if (null != LBMS.getSessions().get(this.clientID).redoUndoable())
{
            return ",cannot-redo;";
        }
        return ",success;";
    }
}

####################
# CloseLibrary.java #
####################

package lbms.command;

/**
 * CloseLibrary class closes the library.
 * @author Team B
 */
public class CloseLibrary implements Command {

    /**
     * Executes the close library command
```

```
     * @return a response
     */
    @Override
    public String execute() {
        return ",library-closed;";
    }

}

####################
# Disconnect.java #
####################

package lbms.command;

import lbms.LBMS;

/**
 * Disconnect class for the disconnect command.
 * @author Team B
 */
public class Disconnect implements Command {

    private long clientID;

    /**
     * Constructor for the Disconnect class.
     * @param clientID: the client to disconnect
     */
    public Disconnect(long clientID) {
        this.clientID = clientID;
    }

    /**
     * Processes the command by interacting with the LBMS models.
     * @return the response string
     */
    @Override
    public String execute() {
        if (LBMS.getSessions().remove(this.clientID) == null) {
            return(",invalid-client-id;");
        }
        return ";";
    }
}

###################################
# MissingParametersException.java #
###################################

package lbms.command;

/**
 * Exception class used when the request given has missing parameters.
 * @author Team B
 */
public class MissingParametersException extends Exception {

    /**
     * Constructor for this exception.
```

```
     * @param message: the message for the exception.
     */
    public MissingParametersException(String message) {
        super(message);
    }

}

########################
# RegisterVisitor.java #
########################

package lbms.command;

import lbms.LBMS;
import lbms.models.PhoneNumber;
import lbms.models.SystemDateTime;
import lbms.models.Visitor;
import lbms.search.UserSearch;

/**
 * RegisterVisitor class that calls the API to register a visitor in the
system.
 * @author Team B
 */
public class RegisterVisitor implements Command {

    private Visitor visitor;

    /**
     * Constructor for the RegisterVisitor command.
     * @param request: the request string to be processed
     * @throws MissingParametersException: missing parameters
     */
    public RegisterVisitor(String request) throws
MissingParametersException {
        String[] arguments = request.split(",");
        if (arguments.length == 1 && arguments[0].equals("")) {
            throw new MissingParametersException("missing-
parameters,{all};");
        } else if (arguments.length == 1) {
            throw new MissingParametersException("missing-
parameters,{last-name,address,phone-number};");
        } else if (arguments.length == 2) {
            throw new MissingParametersException("missing-
parameters,{address,phone-number};");
        } else if (arguments.length == 3) {
            throw new MissingParametersException("missing-
parameters,{phone-number};");
        }

        try {
            this.visitor = new Visitor(arguments[0], arguments[1], null,
null, arguments[2],
                    new PhoneNumber(arguments[3]));
        } catch (ArrayIndexOutOfBoundsException | NumberFormatException
e) {
            throw new MissingParametersException("missing-
parameters,{all};");
        }
```

```
    }

    /**
     * Executes the registration of a visitor.
     * @return the response string or error message
     */
    @Override
    public String execute() {
        if (registerVisitor(this.visitor)) {
            SystemDateTime s = SystemDateTime.getInstance(null);
            return "," + String.format("%010d",
this.visitor.getVisitorID()) + "," +
                    s.getDate().format(SystemDateTime.DATE_FORMAT) + ";";
        }
        return ",duplicate;";
    }

    /**
     * Registers a visitor with the system, if they are not already
registered
     * @param visitor: The visitor to register
     * @return true if successfully registered, false if duplicate
     */
    private static boolean registerVisitor(Visitor visitor) {
        if (UserSearch.BY_ID.findFirst(visitor.getVisitorID()) == null) {
            if (UserSearch.BY_NAME.findFirst(visitor.getName()) == null)
{
                LBMS.getVisitors().put(visitor.getVisitorID(), visitor);
                return true;
            } else {
                Visitor v =
UserSearch.BY_NAME.findFirst(visitor.getName());
                if
(v.getPhoneNumber().toString().equals(visitor.getPhoneNumber().toString()
)) { // uses toString (no .equals)
                    if (v.getAddress().equals(visitor.getAddress())) {
                        return false;
                    } else {
                        LBMS.getVisitors().put(visitor.getVisitorID(),
visitor);
                        return true;
                    }
                }
                LBMS.getVisitors().put(visitor.getVisitorID(), visitor);
                return true;
            }
        }
        return false;
    }
}

######################
# LibrarySearch.java #
######################

package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.models.ISBN;
```

```
import lbms.search.BookSearch;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * LibrarySearch class for the library search command.
 * @author Team B
 */
public class LibrarySearch implements Command {

    private String title, publisher = null, sort_order = null;
    private ArrayList<String> authors;
    private ISBN isbn;
    private long clientID;

    /**
     * Constructor for a LibrarySearch command object.
     * @param request: the request string for a library search
     * @throws MissingParametersException: missing parameters
     */
    public LibrarySearch(long clientID, String request) throws
MissingParametersException {
        this.clientID = clientID;
        String[] arguments = request.split(",");
        if (arguments.length == 0 || arguments.length == 1 &&
arguments[0].equals("")) {
            throw new MissingParametersException("missing-
parameters,title,{authors};");
        }
        if (arguments.length == 1) {
            throw new MissingParametersException("missing-
parameters,{authors};");
        }
        try {
            for (int index = 0; index < arguments.length; index++) {
                if (this.sort_order == null &&
(Arrays.asList(arguments).contains("title") ||
                    Arrays.asList(arguments).contains("publish-date") ||
                    Arrays.asList(arguments).contains("book-status"))) {
                    this.sort_order = arguments[arguments.length - 1];
                }
                if (arguments[index].startsWith("{")) {
                    this.authors = new ArrayList<>();
                    while (!arguments[index].endsWith("}")) {

this.authors.add(arguments[index++].replaceAll("[{}]", ""));
                    }
                    this.authors.add(arguments[index].replaceAll("[{}]",
""));
                } else if (!arguments[index].equals("*")) {
                    if (this.title == null && !arguments[0].equals("*"))
{
                        this.title = (arguments[index]);
                    } else if (this.isbn == null &&
arguments[index].matches("^\\d{13}$")) {
                        this.isbn = new ISBN(arguments[index]);
                    } else if ((this.publisher == null && this.sort_order
== null && index == (arguments.length) - 1) ||
```

```
                                (this.publisher == null && this.sort_order
!= null && index == (arguments.length) - 2)) {
                        this.publisher = arguments[index];
                    }
                }
            }
        } catch (Exception e) {
            throw new MissingParametersException("unknown-error");
        }
    }

    /**
     * Executes the library search command.
     * @return a response string or error message
     */
    @Override
    public String execute() {
        if (this.sort_order != null && !this.sort_order.equals("title")
&& !this.sort_order.equals("publish-date") &&
                !this.sort_order.equals("book-status")) {
            return "invalid-sort-order;";
        }
        List<Book> matches;
        List<Book> antiMatches = new ArrayList<>();
        if (this.title != null) {
            matches =
BookSearch.BY_TITLE.inLibrary().findAll(this.title);
        } else if (this.authors != null) {
            matches =
BookSearch.BY_AUTHOR.inLibrary().findAll(this.authors.get(0));
        } else if (this.isbn != null) {
            matches =
BookSearch.BY_ISBN.inLibrary().findAll(isbn.toString());
        } else if (this.publisher != null) {
            matches =
BookSearch.BY_PUBLISHER.inLibrary().findAll(this.publisher);
        } else {
            matches = new ArrayList<>();
        }

        for (Book b: matches) {
            if (this.title != null &&
!b.getTitle().toLowerCase().contains(this.title.toLowerCase())) {
                antiMatches.add(b);
            }
            if (!LBMS.getBooks().containsKey(b.getIsbn())) {
                antiMatches.add(b);
            }
            if (this.authors != null) {
                for (String author: this.authors) {
                    if (!b.hasAuthorPartial(author)) {
                        antiMatches.add(b);
                    }
                }
            }
            if (this.isbn != null && !b.getIsbn().equals(this.isbn)) {
                antiMatches.add(b);
            }
            if (this.publisher != null &&
!b.getPublisher().toLowerCase().equals(this.publisher.toLowerCase())) {
```

```
                antiMatches.add(b);
            }
        }
        matches.removeAll(antiMatches);

        if (this.sort_order != null) {
            switch (this.sort_order) {
                case "title":
                    matches.sort((Book b1, Book b2) ->
b2.getTitle().compareTo(b1.getTitle()));
                    break;
                case "publish-date":
                    matches.sort((Book b1, Book b2) ->
b2.getPublishDate().compareTo(b1.getPublishDate()));
                    break;
                case "book-status":
                    matches.sort((Book b1, Book b2) ->
                            ((Integer)
b2.getCopiesAvailable()).compareTo(b1.getCopiesAvailable()));
                    break;
            }
        }
        LBMS.getSessions().get(this.clientID).getBookSearch().clear();
        StringBuilder matchesString = new StringBuilder();
        for (Book b: matches) {
            LBMS.getSessions().get(this.clientID).getBookSearch().add(b);
            matchesString.append("\n")
                    .append(b.getCopiesAvailable()).append(",")

.append(LBMS.getSessions().get(this.clientID).getBookSearch().indexOf(b)
+ 1).append(",")
                    .append(b.getIsbn()).append(",")
                    .append(b.getTitle()).append(",{")
                    .append(b.getAuthorsString()).append("},")
                    .append(b.getPublisher()).append(",")
                    .append(b.dateFormat()).append(",")
                    .append(b.getPageCount()).append(",");
        }
        if (matches.size() > 0) {
            matchesString = new StringBuilder(matchesString.substring(0,
matchesString.length() - 1));
        } else {
            return ",0;";
        }

        return "," + matches.size() + "," + matchesString + ";";
    }
}
###################
# BeginVisit.java #
###################

package lbms.command;

import lbms.LBMS;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.models.SystemDateTime;
import lbms.models.Visit;
import lbms.models.Visitor;
import lbms.search.UserSearch;
```

```
/**
 * StartVisit class for the start visit command.
 * @author Team B
 */
public class BeginVisit implements Undoable {

    private long clientID;
    private long visitorID;

    /**
     * Constructor for BeginVisit command.
     * @param request: request string holding clientID and visitorID
     */
    public BeginVisit(String request) {
        String[] arguments = request.split(",");
        if (arguments.length == 1) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID =
LBMS.getSessions().get(this.clientID).getV().getVisitorID();
        } else if (arguments.length == 2) {
            this.clientID = Long.parseLong(arguments[0]);
            this.visitorID = Long.parseLong(arguments[1]);
        }
    }

    /**
     * Executes the BeginVisit command.
     * @return response or error message
     */
    @Override
    public String execute() {

        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-id;";
        }

        if (!ProxyCommandController.assistanceAuthorized(this.visitorID,
this.clientID)) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",not-authorized;";
        }

        Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
        if (UserSearch.BY_ID.findFirst(this.visitorID).getInLibrary()) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",duplicate;";
        }

        Visit v = beginVisit(visitor);
        return "," + String.format("%010d", this.visitorID) + "," +
v.getDate().format(SystemDateTime.DATE_FORMAT)+ ","
                + v.getArrivalTime().format(SystemDateTime.TIME_FORMAT) +
";";
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it failed
```

```
     */
    @Override
    public String unExecute() {
        LBMS.getCurrentVisits().remove(this.visitorID);
        LBMS.getVisitors().get(this.visitorID).switchInLibrary(false);
        return null;
    }

    /**
     * Adds a current visit to the LBMS.
     * @param visitor: the visitor at the library
     * @return the new visit object
     */
    private Visit beginVisit(Visitor visitor) {
        Visit visit = new Visit(visitor);
        LBMS.getCurrentVisits().put(visitor.getVisitorID(), visit);
        return visit;
    }
}
```

```
######################
# SetBookService.java #
######################
```

```
package lbms.command;

import lbms.LBMS;

import static lbms.LBMS.SearchService.GOOGLE;
import static lbms.LBMS.SearchService.LOCAL;

/**
 * SetBookService class for the set book service command.
 * @author Team B
 */
public class SetBookService implements Command {

    private Long clientID;
    private LBMS.SearchService search;

    /**
     * Constructor for a SetBookService class object.
     * @param request: the request string to be processed
     * @throws MissingParametersException: when the request format is
invalid
     */
    public SetBookService(Long clientID, String request) throws
MissingParametersException {
        this.clientID = clientID;
        String s = request.replaceAll(";", "").replaceAll(",", "");
        switch (s) {
            case "local":
                this.search = LOCAL;
                break;
            case "google":
                this.search = GOOGLE;
                break;
            default:
                throw new MissingParametersException("invalid-info-
service;");
```

```
        }
    }

    /**
     * Executes the command by interacting with the backend in the LBMS.
     * @return the response as a string to the system
     */
    @Override
    public String execute() {
        LBMS.getSessions().get(this.clientID).setSearch(this.search);
        return ",success;";
    }
}

####################
# BookPurchase.java #
####################

package lbms.command;

import lbms.LBMS;
import lbms.models.Book;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

/**
 * BookPurchase class that implements the book purchase command.
 * @author Team B
 */
public class BookPurchase implements Undoable {

    private int quantity;
    private List<Integer> ids;
    private long clientID;

    /**
     * Constructor for a BookPurchase class.
     * @param request: the input string
     * @throws MissingParametersException: missing parameters
     */
    public BookPurchase(long clientID, String request) throws
MissingParametersException {
        try {
            this.clientID = clientID;
            ArrayList<String> arguments = new
ArrayList<>(Arrays.asList(request.split(",")));
            this.quantity = Integer.parseInt(arguments.remove(0));
            this.ids =
arguments.parallelStream().map(Integer::parseInt).collect(Collectors.toLi
st());
            if (this.ids.size() == 0) {
                throw new MissingParametersException("missing-
parameters,quantity,id[,ids];");
            }
        } catch (Exception e) {
            throw new MissingParametersException("missing-
parameters,quantity,id[,ids];");
```

```
        }
    }

    /**
     * Executes the book purchase command.
     * @return a success message for the command
     */
    @Override
    public String execute() {
        if (this.ids.size() == 0) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",missing-parameters,id;";
        }
        String s = processPurchaseOrder();
        if (s.equals(",failure;")) {
            LBMS.getSessions().get(clientID).popUndoable();
            return s;
        }
        s = s.replaceAll(",$","");
        return ",success," + s + ";";
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it failed
     */
    @Override
    public String unExecute() {
        for (int id: this.ids) {
            Book b;
            b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
            for (int i = 0; i < this.quantity; i++) {
                b.undoPurchase();
            }
            if (LBMS.getBooks().get(b.getIsbn()).getNumberOfCopies() <=
0) {
                LBMS.getBooks().remove(b.getIsbn());
            }
        }
        return null;
    }

    /**
     * Buys *quantity* of each book listed in *ids*
     * @return a response string
     */
    private String processPurchaseOrder() {
        String booksBought = "";
        for (int id: this.ids) {
            Book b;

            try {
                b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
            } catch (IndexOutOfBoundsException e) {
                return ",failure;";
            }

            for (int i = 0; i < this.quantity; i++) {
```

```
                buyBook(b);
            }

            booksBought += ("\n" + b.toString() + "," + this.quantity) +
","; 
        }
        return this.ids.size() + booksBought;
    }

    /**
     * Buys a book for the library
     * @param book: The book to buy
     */
    private void buyBook(Book book) {
        book.purchase();
        if (!LBMS.getBooks().values().contains(book)) {
            LBMS.getBooks().put(book.getIsbn(), book);
        }
    }
}

###############
# Borrow.java #
###############

package lbms.command;

import lbms.LBMS;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.models.Book;
import lbms.models.SystemDateTime;
import lbms.models.Transaction;
import lbms.models.Visitor;
import lbms.search.UserSearch;

import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * Borrow class that implements the borrow command.
 * @author Team B
 */
public class Borrow implements Undoable {

    private long clientID;
    private long visitorID;
    private ArrayList<Integer> ids = new ArrayList<>();

    /**
     * Constructor for a Borrow class.
     * @param request: the request input string
     * @throws MissingParametersException: missing parameters
     */
    public Borrow(String request) throws MissingParametersException {
        String[] allArguments = request.split(",");
        if (allArguments.length < 2) {
            throw new MissingParametersException(",missing-
parameters,{ids};");
```

```
        }

        this.clientID = Long.parseLong(allArguments[0]);
        String[] arguments = Arrays.copyOfRange(allArguments, 1,
allArguments.length);

        for (String arg: arguments) {
            if (arg.equals(arguments[arguments.length - 1]) &&
!arg.endsWith("}")) {
                throw new MissingParametersException(",missing-
parameters,{ids};");
            } else if (arg.endsWith("}")) {
                break;
            }
        }

        if (arguments[arguments.length-1].startsWith("{") &&
arguments[arguments.length-1].endsWith("}")) {
            this.ids.add(Integer.parseInt(arguments[arguments.length-
1].replaceAll("[{}]","")));
        } else if (arguments[arguments.length-1].endsWith("}")) {
            for (String arg: arguments) {
                if (arg.startsWith("{") || arg.endsWith("}")) {
                    this.ids.add(Integer.parseInt(arg.replaceAll("[{}]",
"")));
                } else {
                    this.ids.add(Integer.parseInt(arg));
                }
            }
        } else {
            for (int i = 0; i < arguments.length - 1; i++) {
                if (arguments[i].startsWith("{") ||
arguments[i].endsWith("}")) {

this.ids.add(Integer.parseInt(arguments[i].replaceAll("[{}]", "")));
                    if (arguments[i].endsWith("}")) {
                        break;
                    }
                } else {
                    this.ids.add(Integer.parseInt(arguments[i]));
                }
            }
        }

        if (arguments[arguments.length - 1].endsWith("}")) {
            this.visitorID =
LBMS.getSessions().get(this.clientID).getV().getVisitorID();
        } else {
            this.visitorID = Long.parseLong(arguments[arguments.length -
1]);
        }
    }

    /**
     * Executes the borrow command.
     * @return the response or error message
     */
    @Override
    public String execute() {
```

```
        if (!ProxyCommandController.assistanceAuthorized(this.visitorID,
this.clientID)) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",not-authorized;";
        }

        if (UserSearch.BY_ID.findFirst(this.visitorID) == null) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",invalid-visitor-id;";
        } else if (UserSearch.BY_ID.findFirst(this.visitorID).getFines()
> 0) {
            LBMS.getSessions().get(clientID).popUndoable();
            return ",outstanding-fine," + new
DecimalFormat("#.00").format(UserSearch.BY_ID
                    .findFirst(this.visitorID).getFines()) + ";";
        }
        StringBuilder invalidIDs = new StringBuilder();
        String temp = "";
        for (Integer i: this.ids) {
            if
(!UserSearch.BY_ID.findFirst(this.visitorID).canCheckOut()) {
                LBMS.getSessions().get(clientID).popUndoable();
                return ",book-limit-exceeded;";
            }
            if (i <=
LBMS.getSessions().get(this.clientID).getBookSearch().size() &&

LBMS.getSessions().get(this.clientID).getBookSearch().get(i -
1).getCopiesAvailable() < 1) {
                LBMS.getSessions().get(clientID).popUndoable();
                return ",no-more-copies;";
            }
            temp = checkOutBook(i, this.visitorID);
            try {
                if (temp.contains("id-error")) {
                    String[] error = temp.split(",");
                    invalidIDs.append(error[1]).append(",");
                }
            } catch (NullPointerException e) {
                e.printStackTrace();
                System.exit(1);
            }
        }
        if (invalidIDs.length() > 0) {
            String output = "invalid-book-id,";
            LBMS.getSessions().get(clientID).popUndoable();
            invalidIDs.deleteCharAt(invalidIDs.length()-1);
            output += "{" + invalidIDs + "}";
            //output = output.substring(0,output.length() - 1);
            output += ";";
            return "," + output;
        } else {
            return "," + temp + ";";
        }
    }

    /**
     * Un-executes the command.
     * @return null if successful, a string if it fails
     */
```

```
    @Override
    public String unExecute() {
        for (int id: this.ids) {
            Book b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
            Transaction t = new Transaction(b.getIsbn(), this.visitorID);
            Visitor visitor = UserSearch.BY_ID.findFirst(this.visitorID);
            visitor.undoCheckOut(t);
            b.undoCheckOut();
            List<Transaction> transactions = LBMS.getTransactions();
            transactions.remove(t);
        }
        return null;
    }

    /**
     * Checks out a book for a visitor.
     * @param id: the temp id of the book
     * @param visitorID: the ID of the visitor checking out the book
     * @return a string of the response message
     */
    private String checkOutBook(int id, long visitorID) {
        Book b;
        Visitor v;
        Transaction t;
        try {
            b =
LBMS.getSessions().get(this.clientID).getBookSearch().get(id - 1);
            t = new Transaction(b.getIsbn(), visitorID);
            v = UserSearch.BY_ID.findFirst(visitorID);
        } catch (Exception e) {
            LBMS.getSessions().get(clientID).popUndoable();
            return "id-error," + id;
        }

        if (v.canCheckOut()) {
            v.checkOut(t);
            b.checkOut();
            List<Transaction> transactions = LBMS.getTransactions();
            transactions.add(t);
            return t.getDueDate().format(SystemDateTime.DATE_FORMAT);
        }
        LBMS.getSessions().get(clientID).popUndoable();
        return "unknown-error";
    }
}

#########################
# StatisticsReport.java #
#########################

package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.models.SystemDateTime;
import lbms.models.Visit;
import lbms.models.Visitor;

import java.time.Duration;
```

```
import java.time.LocalDate;
import java.util.ArrayList;

/**
 * StatisticsReport class implements the statistics report command.
 * @author Team B
 */
public class StatisticsReport implements Command {

    private Integer days;

    /**
     * Constructor for a StatisticsReport command.
     * @param request: the request string to be processed
     */
    public StatisticsReport(String request) throws
MissingParametersException {
        try {
            if (!request.equals("")) {
                this.days = Integer.parseInt(request);
            }
        } catch (NumberFormatException e) {
            throw new MissingParametersException("incorrect-value-for-
days;");
        }
    }

    /**
     * Executes the command on the system.
     * @return a string of the response
     */
    @Override
    public String execute() {
        return "," +
SystemDateTime.getInstance(null).getDate().format(SystemDateTime.DATE_FOR
MAT) + ",\n" +
                generateReport(this.days);
    }

    /**
     * Generates a Library report including the following information:
     *     -total number of books in the library
     *     -total number of registered library visitors
     *     -average length of a visit (hh:mm:ss)
     *     -number of books purchased
     *     -amount of fines collected
     *     -amount of fines outstanding
     * @param days: the number of days that the report should include in
its statistics
     *            if null: report should include statistics using all
data
     * @return a string of the response message
     */
    private String generateReport(Integer days) {
        String report = "";
        Duration totalVisitTime = Duration.ZERO;
        Duration averageVisitTime = Duration.ZERO;
        int booksPurchased = LBMS.getBooks().size();
        double collectedFines = 0;
        double outstandingFines = 0;
```

```
        //calculate total outstanding fines
        for (Visitor v: LBMS.getVisitors().values()) {
            outstandingFines += v.getFines();
        }

        //calculate payed fines
        for (Visitor v: LBMS.getVisitors().values()) {
            collectedFines += v.getPayedFines();
        }

        if (days != null) {

            LocalDate reportStartDate =
SystemDateTime.getInstance(null).getDate().minusDays(days);
            LocalDate reportEndDate =
SystemDateTime.getInstance(null).getDate();

            // grabbing relevant visits
            ArrayList<Visit> visitsInReport = new ArrayList<>();
            for (Visit v: LBMS.getTotalVisits()) {
                if (v.getDate().isBefore(reportEndDate) &&
v.getDate().isAfter(reportStartDate)) {
                    visitsInReport.add(v);
                }
            }
            // calculating average visit time for all visits in system
            for (Visit v: visitsInReport) {
                totalVisitTime.plus(v.getDuration());
            }
            if (visitsInReport.size() != 0) {
                averageVisitTime =
totalVisitTime.dividedBy(visitsInReport.size());
            }

            // determine number of books purchased in timeframe
            booksPurchased = 0;
            for (Book b: LBMS.getBooks().values()) {
                if (b.getPurchaseDate().isBefore(reportEndDate) &&
b.getPurchaseDate().isAfter(reportStartDate)) {
                    booksPurchased++;
                }
            }
        } else {
            // calculating average visit time for all visits in system
            for (Visit v : LBMS.getTotalVisits()) {
                totalVisitTime.plus(v.getDuration());
            }
            if (LBMS.getTotalVisits().size() != 0) {
                averageVisitTime =
totalVisitTime.dividedBy(LBMS.getTotalVisits().size());
            }
        }

        report += ("Number of Books: " + LBMS.getBooks().size() + "\n" +
                "Number of Visitors: " + LBMS.getVisitors().size() + "\n"
+
                "Average Length of Visit: " +
formatDuration(averageVisitTime) + "\n" +
```

```
                "Number of Books Purchased: " + booksPurchased + "\n" +
                "Fines Collected: " + collectedFines + "\n" +
                "Fines Outstanding: " + outstandingFines);

        return report + ";";
    }

    /**
     * Formats the durations.
     * @param duration: the duration to be formatted
     * @return a string of the formatted duration
     */
    private static String formatDuration(Duration duration) {
        long s = duration.getSeconds();
        return String.format("%02d:%02d:%02d", s / 3600, (s % 3600) / 60,
(s % 60));
    }
}

#############
# Undo.java #
#############

package lbms.command;

import lbms.LBMS;

/**
 * Undo class for the undo command.
 * @author Team B
 */
public class Undo implements Command {

    private Long clientID;

    /**
     * Constructor for an Undo class object.
     * @param clientID: the id of the client to undo
     */
    public Undo(Long clientID) {
        this.clientID = clientID;
    }

    /**
     * Executes the command by interacting with the backend in the LBMS.
     * @return the response as a string to the system
     */
    @Override
    public String execute() {
        if (null != LBMS.getSessions().get(this.clientID).undoUndoable())
{
            return ",cannot-undo;";
        }
        return ",success;";
    }
}

####################
# AdvanceTime.java #
####################
```

```
package lbms.command;

import lbms.models.SystemDateTime;

/**
 * AdvanceTime class that calls the API to advance system time.
 * @author Team B
 */
public class AdvanceTime implements Command {

    private long days;
    private long hours;

    /**
     * Constructor for AdvanceTime class.
     * @param request: the input string of the request
     */
    public AdvanceTime(String request) {
        String[] arguments = request.split(",");
        this.days = Long.parseLong(arguments[0]);
        if (arguments.length > 1) {
            this.hours = Long.parseLong(arguments[1]);
        } else {
            this.hours = 0L;
        }
    }

    /**
     * Executes the advance time command.
     * @return the response or error message
     */
    @Override
    public String execute() {
        if (this.days < 0 || this.days > 7) {
            return ",invalid-number-of-days," + this.days + ";";
        }
        if (this.hours < 0 || this.hours > 23) {
            return ",invalid-number-of-hours," + this.hours + ";";
        }
        if (this.hours == 0 && this.days == 0) {
            return ",invalid-number-of-hours," + this.hours + ";";
        }
        SystemDateTime.getInstance(null).plusDays(this.days);
        SystemDateTime.getInstance(null).plusHours(this.hours);
        return ",success;";
    }
}

########################
# GoogleAPISearch.java #
########################

package lbms.search;

import com.google.gson.*;
import com.google.gson.reflect.TypeToken;
import lbms.models.Book;
import lbms.models.ISBN;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.lang.reflect.Type;
import java.net.HttpURLConnection;
import java.net.URL;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;

/**
 * Searches the Google Books API
 * @author Team B
 */
public final class GoogleAPISearch {

    private static final String baseURL =
"https://www.googleapis.com/books/v1/volumes?q=";
    private static final String saleableSuffix = "&filter=paid-ebooks";
    private static final String maxResultsSuffix = "&maxResults=20";

    /**
     * Searches the Google Books API by book title.
     * @param title string representing the title of the book
     * @return a list of book objects matching the search parameter
     */
    public static List<Book> searchByTitle(String title) {
        return parseJSON(query(baseURL + "intitle:\"" +
title.replaceAll(" ", "+") + "\"" + saleableSuffix +
                maxResultsSuffix));
    }

    /**
     * Searches the Google Books API by author(s).
     * @param authors ArrayList of authors
     * @return a list of book objects matching the search parameter
     */
    public static List<Book> searchByAuthor(ArrayList<String> authors) {
        String authorString = "";
        for (String author : authors) {
            authorString += ("\"" + author + "\"");
        }
        String formattedAuthors = authorString.replaceAll(" ",
"+").replaceAll("\"\"", "\"+\"");

        return parseJSON(query(baseURL + "inauthor:" + formattedAuthors +
saleableSuffix + maxResultsSuffix));
    }

    /**
     * Searches the Google Books API by ISBN.
     * @param isbn string representing the ISBN of the book.
     * @return a list of book objects matching the search parameter
     */
    public static List<Book> searchByISBN(String isbn) {
        return parseJSON(query(baseURL + "isbn:" + isbn + saleableSuffix
+ maxResultsSuffix));
```

```
    }

    /**
     * Searches the Google Books API by publisher.
     * @param publisher string representing the publisher of the book
     * @return a list of book objects matching the search parameter
     */
    public static List<Book> searchByPublisher(String publisher) {
        return parseJSON(query(baseURL + "inpublisher:\"" +
publisher.replaceAll(" ", "+" + "\"") + saleableSuffix
                + maxResultsSuffix));
    }

    /**
     * Uses the URL to gather the JSON data from Google books
     * @param url the complete URL to perform the GET  request
     * @return a String representation of the JSON
     */
    private static String query(String url) {
        String responseString = "";

        try {
            URL GoogleURL = new URL(url);
            HttpURLConnection con = (HttpURLConnection)
GoogleURL.openConnection();
            con.setRequestMethod("GET");
            con.setConnectTimeout(10000);
            con.setReadTimeout(10000);

            BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }

            in.close();

            responseString = response.toString();

        } catch (IOException e) {
            System.out.println("Improper Google API Query");
        }

        return responseString;
    }

    /**
     * Parses the JSON into a List of Book objects
     * @param response a String representation of the JSON
     * @return a list of book objects matching the search parameter
     */
    private static List<Book> parseJSON(String response){
        List<Book> results = new ArrayList<>();

        JsonElement jelement = new JsonParser().parse(response);
        JsonObject jobject = jelement.getAsJsonObject();
        JsonArray books = jobject.getAsJsonArray("items");
```

```
        for (int i = 0; i < books.size(); i++) {
            try {
                JsonObject book = books.get(i).getAsJsonObject();
                JsonObject volumeInfo =
book.get("volumeInfo").getAsJsonObject();

                //getting title
                String title;
                if (volumeInfo.get("title") == null) {
                    continue; // skip book
                } else {
                    title = volumeInfo.get("title").toString();
                }

                // getting publisher
                String publisher;
                if (volumeInfo.get("publisher") == null) {
                    continue; // skip book
                } else {
                    publisher = volumeInfo.get("publisher").toString();
                }

                // getting pageCount
                int pageCount;
                if (volumeInfo.get("pageCount") == null) {
                    pageCount = 0;
                } else {
                    pageCount =
Integer.parseInt(volumeInfo.get("pageCount").toString());
                }

                // getting ISBN
                ISBN isbn;
                if (volumeInfo.get("industryIdentifiers") == null) {
                    continue; // skip book
                } else {
                    JsonObject isbnInfo =

volumeInfo.get("industryIdentifiers").getAsJsonArray().get(0).getAsJsonOb
ject();
                    isbn = new
ISBN(isbnInfo.get("identifier").toString().replaceAll("\"", ""));
                }

                // getting publishDate
                Calendar publishDate = Calendar.getInstance();
                if (volumeInfo.get("publishedDate") == null) {
                    continue; // skips book
                } else {
                    SimpleDateFormat format = new SimpleDateFormat("yyyy-
MM-dd");
                    SimpleDateFormat formatNoDay = new
SimpleDateFormat("yyyy-MM");
                    SimpleDateFormat formatOnlyYear = new
SimpleDateFormat("yyyy");
                    String publishedDateString =
volumeInfo.get("publishedDate").toString().replaceAll("\"", "");
                    if (publishedDateString.length() == 4) {
                        publishedDateString += "-01-01";
```

```
                    }
                    Date date;
                    if (publishedDateString.length() > 7) {
                        date = format.parse(publishedDateString);
                    } else if (publishedDateString.length() > 4){
                        date = formatNoDay.parse(publishedDateString);
                    } else {
                        date = formatOnlyYear.parse(publishedDateString);
                    }

                    publishDate.setTime(date);
                }

                // getting authors
                ArrayList<String> authors;
                if (volumeInfo.get("authors") == null) {
                    continue; // skip book
                } else {
                    Gson converter = new Gson();
                    Type type = new
TypeToken<List<String>>(){}.getType();
                    authors =
converter.fromJson(volumeInfo.get("authors").getAsJsonArray(), type );
                }


                // create Book object
                results.add(new Book(isbn, title, authors, publisher,
publishDate, pageCount, 0, 0));

            } catch (ParseException e) {
                System.out.println("Improper JSON parse");
            }
        }

        return results;
    }
}

###################
# BookSearch.java #
###################

package lbms.search;

import lbms.LBMS;
import lbms.models.Book;

import java.util.Collection;
import java.util.function.Predicate;
import java.util.stream.Stream;

/**
 * Searches the books.
 * @author Team B
 */
public enum BookSearch implements Search<Book> {
    BY_AUTHOR,
    BY_ISBN,
    BY_TITLE,
```

```
    BY_PUBLISHER;

    private Collection<Book> toSearch = LBMS.getBooks().values();

    /**
     * Creates a predicate for the search.
     * @param s: the string used for the search
     * @return the predicate that was created
     */
    @Override
    public Predicate<? super Book> createPredicate(String s) {
        switch (this) {
            case BY_AUTHOR:
                return book -> book.hasAuthorPartial(s);
            case BY_ISBN:
                return book -> book.getIsbn().toString().contains(s);
            case BY_TITLE:
                return book ->
book.getTitle().toLowerCase().contains(s.toLowerCase());
            case BY_PUBLISHER:
                return book ->
book.getPublisher().toLowerCase().contains(s.toLowerCase());
            default:
                return book -> true;
        }
    }

    /**
     * Creates a stream that is filtered by the given condition
predicate.
     * @param condition: the predicate for the filter
     * @return a stream of books
     */
    @Override
    public Stream<Book> filterStream(Predicate<? super Book> condition) {
        return toSearch.parallelStream().filter(condition);
    }

    /**
     * Creates an instance of this enum.
     * @return a BookSearch enum object.
     */
    public BookSearch toBuy() {
        toSearch = LBMS.getBooksToBuy();
        return this;
    }

    /**
     * Creates an instance of this enum.
     * @return a BookSearch enum object for searching books already in
the libary.
     */
    public BookSearch inLibrary() {
        toSearch = LBMS.getBooks().values();
        return this;
    }
}

###################
# UserSearch.java #
```

```
##################

package lbms.search;

import lbms.LBMS;
import lbms.models.Visitor;

import java.util.function.Predicate;
import java.util.stream.Stream;

/**
 * UserSearch class finds users in the system.
 * @author Team B
 */
public enum UserSearch implements Search<Visitor> {
    BY_ID,
    BY_NAME,
    BY_ADDRESS,
    BY_PHONE;

    /**
     * Creates a predicate condition for the stream.
     * @param s: the condition for the predicate
     * @return a predicate condition
     */
    @Override
    public Predicate<? super Visitor> createPredicate(String s) {
        switch (this) {
            case BY_ID:
                return visitor ->
Long.toString(visitor.getVisitorID()).contains(s);
            case BY_NAME:
                return visitor -> visitor.getName().contains(s);
            case BY_ADDRESS:
                return visitor -> visitor.getAddress().contains(s);
            case BY_PHONE:
                return visitor ->
visitor.getPhoneNumber().toString().contains(s);
            default:
                return visitor -> true;
        }
    }

    /**
     * Searches the users in the Library Book Management System.
     * @param condition: the condition for the stream
     * @return a stream of visitors that match the condition
     */
    @Override
    public Stream<Visitor> filterStream(Predicate<? super Visitor>
condition) {
        return
LBMS.getVisitors().values().parallelStream().filter(condition);
    }
}

###############
# Search.java #
###############
```

```
package lbms.search;

import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;
import java.util.stream.Stream;

/**
 * Interface to model search classes on.
 * @author Team B
 */
public interface Search<T> {

    /**
     * Creates the predicate for searching.
     * @param s: the condition for the predicate
     * @return a predicate with the given condition
     */
    Predicate<? super T> createPredicate(String s);

    /**
     * Creates a filtered stream with the given condition.
     * @param condition: the condition for the stream
     * @return a stream
     */
    Stream<T> filterStream(Predicate<? super T> condition);

    /**
     * Finds all the objects from a given search.
     * @param s: the predicate condition
     * @return a list of the results
     */
    default List<T> findAll(String s) {
        return
filterStream(createPredicate(s)).collect(Collectors.toList());
    }

    /**
     * Searches for the first match, this method calls the other
findFirst that takes a string.
     * @param l: the long to be converted to a string
     * @return the first matching object
     */
    default T findFirst(Long l) {
        return findFirst(l.toString());
    }

    /**
     * Finds the first match for the search.
     * @param s: the predicate condition
     * @return the first matching object
     */
    default T findFirst(String s) {
        return filterStream(createPredicate(s)).findFirst().orElse(null);
    }

}

#############
# ISBN.java #
```

############

```
package lbms.models;

import java.io.Serializable;

/**
 * ISBN class used to store the isbn for a book.
 * @author Team B
 */
public class ISBN implements Serializable {

    private static final long serialVersionUID = 1L;
    private final String value;

    /**
     * Constructor for an ISBN
     * @param isbn: the isbn to be set
     */
    public ISBN(String isbn) {
        this.value = isbn.replaceAll("-", "").trim();
    }

    /**
     * Creates a string of the isbn.
     * @return the string representation of the isbn
     */
    @Override
    public String toString() {
        return this.value;
    }

    /**
     * Determines if the isbn is valid or not.
     * @return true if valid, false if not
     */
    boolean isValid() {
        switch (this.value.length()) {
            case 10:
                return isValidISBN10();
            case 13:
                return isValidISBN13();
            default:
                return false;
        }
    }

    /**
     * Determines if a 10 digit isbn is valid.
     * @return true if valid, false if not
     */
    private boolean isValidISBN10() {
        int sum = 0;
        int n;
        String s;

        for (int i = 10; i > 0; i--) {
            s = this.value.substring(10 - i, 10 - i + 1);
            if (s.equals("X") || s.equals("x")) {
                n = 10;
```

```
            } else {
                try {
                    n = Integer.parseInt(s);
                } catch (NumberFormatException e) {
                    return false;
                }
            }
            sum += i * n;
        }

        return (sum % 11 == 0);
    }

    /**
     * Determines if a 13 digit isbn is valid.
     * @return true if valid, false if not
     */
    private boolean isValidISBN13() {
        int sum = 0;
        int n;
        int m;
        String s;

        for (int i = 1; i < 13; i++) {
            s = this.value.substring(i - 1, i);
            try {
                n = Integer.parseInt(s);
            } catch (NumberFormatException e) {
                return false;
            }

            m = ((i % 2 == 1) ? 1 : 3);
            sum += n * m;
        }

        try {
            n = Integer.parseInt(this.value.substring(12, 13));
        } catch (NumberFormatException e) {
            return false;
        }

        return (((10 - sum % 10) % 10) - n == 0);
    }

    /**
     * Used to determine if two ISBNs are equal
     * @param o: the other isbn
     * @return true if equal, false if not
     */
    @Override
    public boolean equals(Object o) {
        if (!(o instanceof ISBN)) return false;

        ISBN isbn = (ISBN)o;
        return this.value.equals(isbn.toString());
    }
}

####################
# ClosedState.java #
```

```
####################

package lbms.models;

/**
 * ClosedState class used for the status of the library when it is
closed.
 * @author Team B
 */
public class ClosedState implements LibraryState {

    /**
     * Returns false because when the library is in this state it is
always closed.
     * @return false
     */
    public boolean isOpen() {
        return false;
    }

}

#############
# Book.java #
#############

package lbms.models;

import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.time.LocalDate;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.stream.Collectors;

/**
 * Class for a Book object, used in the library book management system.
 * @author Team B
 */
public class Book implements Serializable, Comparable<Book> {

    private String title, publisher;
    private ArrayList<String> authors;
    private ISBN isbn;
    private int pageCount, numberOfCopies, copiesCheckedOut;
    private Calendar publishDate;
    private LocalDate purchaseDate;

    /**
     * Constructor for a Book.
     * @param isbn: the isbn number
     * @param title: the title of the book
     * @param authors: the list of authors of the book
     * @param publisher: the publisher of the book
     * @param publishDate: the date the book was published
     * @param pageCount: the number of pages in the book
     * @param numberOfCopies: the quantity of this book the library owns
     * @param copiesCheckedOut: the total number of books that are not
available
     */
```

```
    public Book(ISBN isbn, String title, ArrayList<String> authors,
String publisher, Calendar publishDate,
                int pageCount, int numberOfCopies, int copiesCheckedOut)
{
        this.isbn = isbn;
        this.title = title;
        this.authors = authors;
        this.publisher = publisher;
        this.publishDate = publishDate;
        this.pageCount = pageCount;
        this.numberOfCopies = numberOfCopies;
        this.copiesCheckedOut = copiesCheckedOut;
    }

    /**
     * Getter for the title.
     * @return the title of the book
     */
    public String getTitle() {
        return this.title;
    }

    /**
     * Getter for the page count
     * @return the page count
     */
    public int getPageCount() {
        return this.pageCount;
    }

    /**
     * Getter for the publisher.
     * @return the publisher of the book
     */
    public String getPublisher() {
        return this.publisher;
    }

    /**
     * Getter for the authors.
     * @return the list of authors of the book
     */
    public ArrayList<String> getAuthors() {
        return this.authors;
    }

    /**
     * Determines if the string is a partial author.
     * @param name: the author name
     * @return true if it is a partial author
     */
    public boolean hasAuthorPartial(String name) {
        return !getAuthors().parallelStream().filter(author ->
author.toLowerCase().contains(name.toLowerCase()))
                .collect(Collectors.toList()).isEmpty();
    }

    /**
     * Getter for the ISBN.
     * @return the ISBN number of the book
```

```
 */
public ISBN getIsbn() {
    return this.isbn;
}

/**
 * Getter for the number of copies.
 * @return the quantity of this book the library owns
 */
public int getNumberOfCopies() {
    return this.numberOfCopies;
}

/**
 * Calculates the number of copies currently available.
 * @return the number of copies of this book that are available
 */
public int getCopiesAvailable() {
    return this.numberOfCopies - this.copiesCheckedOut;
}

/**
 * Getter for the published date.
 * @return the publishing date for the book
 */
public Calendar getPublishDate() {
    return this.publishDate;
}

/**
 * Getter for the purchase date.
 * @return the latest date of purchase (desired?)
 */
public LocalDate getPurchaseDate() {
    return this.purchaseDate;
}

/**
 * Checks out a book.
 */
public void checkOut() {
    if (this.copiesCheckedOut < this.numberOfCopies) {
        this.copiesCheckedOut++;
    }
}

/**
 * Undoes the checkout of a book.
 */
public void undoCheckOut() {
    this.copiesCheckedOut--;
}

/**
 * Returns a book.
 */
public void returnBook() {
    this.copiesCheckedOut--;
}
```

```
/**
 * Undoes the return of a book.
 */
public void undoReturnBook() {
    this.copiesCheckedOut++;
}

/**
 * String formatting used in API method to purchase books.
 * @return a string representation of the book in a specific format
 */
@Override
public String toString() {
    String output = this.isbn + "," + this.title + ",{";
    for (String author: this.authors) {
        output += author + ",";
    }
    output = output.substring(0, output.length() - 1);
    output += "}," + dateFormat();
    return output;
}

/**
 * Formats the calendar date to a string format.
 * @return a string of the published date
 */
public String dateFormat() {
    SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");
    return sdf.format(this.publishDate.getTime());
}

/**
 * Compares a book to this instance of a book.
 * @param book: the book to be compared to
 * @return int representing the comparison of the book titles
 */
@Override
public int compareTo(Book book) {
    String compareTitle = book.getTitle();
    return this.title.compareTo(compareTitle);
}

/**
 * Sets the purchase date when a book is purchased.
 */
public void purchase() {
    this.purchaseDate = SystemDateTime.getInstance(null).getDate();
    this.numberOfCopies++;
}

/**
 * Undoes the purchase of a book.
 */
public void undoPurchase() {
    this.numberOfCopies--;
}

/**
 * Get all authors as a string
 * @return a string representation of all authors
```

```
     */
    public String getAuthorsString() {
        String authors = "";
        for (String author: this.authors) {
            authors += author + ",";
        }
        return authors.replaceAll(",$", "");
    }
}

################
# Session.java #
################

package lbms.models;

import lbms.LBMS;
import lbms.LBMS.SearchService;
import lbms.command.Undoable;

import java.util.ArrayList;
import java.util.Stack;

import static lbms.LBMS.SearchService.LOCAL;

/**
 * Session class for the LBMS
 * @author Team B
 */
public class Session {

    private long clientID;
    private Visitor v;
    private SearchService search;
    private Stack<Undoable> undoStack;
    private Stack<Undoable> redoStack;
    private ArrayList<Book> bookSearch;

    /**
     * Constructor for a Session.
     */
    public Session() {
        LBMS.incrementSessions();
        this.clientID = LBMS.getTotalSessions();
        this.v = null;
        this.search = LOCAL;
        this.undoStack = new Stack<>();
        this.redoStack = new Stack<>();
        this.bookSearch = new ArrayList<>();
    }

    /**
     * Getter for the clientID.
     * @return the clientID
     */
    public long getClientID() {
        return this.clientID;
    }

    /**
```

```
 * Setter for the visitor.
 * @param v: the visitor to be set
 */
public void setV(Visitor v) {
    this.v = v;
}

/**
 * Getter for the visitor.
 * @return the visitor for the session
 */
public Visitor getV() {
    return this.v;
}

/**
 * Adds an undoable command to the undostack.
 * @param u: the undoable command to be added
 */
public void addUndoable(Undoable u) {
    this.undoStack.push(u);
}

public void popUndoable() {
    this.undoStack.pop();
}

/**
 * Pops an undoable off the stack and un-executes it.
 * @return failure message or null if success
 */
public String undoUndoable() {
    Undoable u;
    if (this.undoStack.size() > 0) {
        u = this.undoStack.pop();
    } else {
        return "failure";
    }
    u.unExecute();
    this.redoStack.push(u);
    return null;
}

/**
 * Pops an undoable off the redo stack and executes it.
 * @return failure message or null if success
 */
public String redoUndoable() {
    Undoable u;
    if (this.redoStack.size() > 0) {
        u = this.redoStack.pop();
    } else {
        return "failure";
    }
    u.execute();
    this.undoStack.push(u);
    return null;
}

/**
```

```
     * Sets the search service for the session.
     * @param ss: the search service to be set
     */
    public void setSearch(SearchService ss) {
        this.search = ss;
    }

    /**
     * Getter for the search service.
     * @return the search type being used
     */
    public SearchService getSearch() {
        return this.search;
    }

    /**
     * Clears the undo and redo stacks.
     */
    public void clearStacks() {
        this.undoStack.clear();
        this.redoStack.clear();
    }

    /**
     * Getter for the book search ArrayList.
     * @return the list of books from the search
     */
    public ArrayList<Book> getBookSearch() {
        return this.bookSearch;
    }
}

####################
# Transaction.java #
####################

package lbms.models;

import java.io.Serializable;
import java.time.LocalDate;
import java.time.Period;

/**
 * Class for a Transaction object, used in the library book management
system.
 * @author Team B
 */
public class Transaction implements Serializable {

    /** Constants for overdue fines. */
    private final static double MAX_FINE = 30.00;
    private final static double WEEK_FINE = 2.00;
    private final static double INITIAL_FINE = 10.00;

    private ISBN isbn;
    private long visitorId;
    private LocalDate date, dueDate, closeDate;

    /**
     * Constructor for a Transaction object.
```

```
     * @param isbn: the isbn of the book
     * @param visitorId: the ID of the visitor checking it out
     */
    public Transaction(ISBN isbn, long visitorId) {
        this.isbn = isbn;
        this.visitorId = visitorId;
        this.date = SystemDateTime.getInstance(null).getDate();
        this.dueDate = date.plusDays(7);
    }

    /**
     * Getter for the ISBN number.
     * @return the isbn of the book checked out
     */
    public ISBN getIsbn() {
        return this.isbn;
    }

    /**
     * Getter for the visitors ID.
     * @return the visitors ID
     */
    public long getVisitor() {
        return this.visitorId;
    }

    /**
     * Getter for the fine.
     * @return the fine due on the book
     */
    double getFine() {
        int days = Period.between(this.dueDate,
SystemDateTime.getInstance(null).getDate()).getDays();
        double fine = 0.0;
        for (int i = 0; i < days; i++) {
            if (i == 0) {
                fine += INITIAL_FINE;
            } else {
                fine += WEEK_FINE;
            }
        }
        if (fine < MAX_FINE) {
            return fine;
        }
        return MAX_FINE;
    }

    /**
     * Marks that the fine has been paid for this transaction
     */
    public void closeTransaction() {
        this.closeDate = SystemDateTime.getInstance(null).getDate();
    }

    /**
     * Getter for the date.
     * @return the date the book was checked out
     */
    public LocalDate getDate() {
        return this.date;
```

```
    }

    /**
     * Getter for the date the book is due.
     * @return the date the book is due
     */
    public LocalDate getDueDate() {
        return this.dueDate;
    }
}

################
# Visitor.java #
################

package lbms.models;

import lbms.LBMS;

import java.io.Serializable;
import java.util.HashMap;

/**
 * Class for a Visitor object, used in the library book management
system.
 * @author Team B
 */
public class Visitor implements Serializable {

    private String firstName, lastName;
    private String username;
    private String password;
    private String address;
    private PhoneNumber phoneNumber;
    private long visitorID;
    private HashMap<ISBN, Transaction> checkedOutBooks;
    private HashMap<ISBN, Transaction> previousCheckedOutBooks;
    private final int MAX_BOOKS = 5;
    private boolean inLibrary;
    private double currentFines;
    private double totalFines;
    private double payedFines;

    /**
     * Constructor for a Visitor object.
     * @param firstName: the visitor's first name
     * @param lastName: the last name of the visitor
     * @param username: the visitor's username, if they have an account
     * @param password: the visitor's password, if they have an account
     * @param address: the visitor's address
     * @param phoneNumber: the visitor's phone number
     */
    public Visitor(String firstName, String lastName, String username,
String password, String address,
                   PhoneNumber phoneNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.username = username;
        this.password = password;
        this.address = address;
```

```
        this.phoneNumber = phoneNumber;
        this.visitorID = LBMS.getVisitors().size() + 1;
        this.checkedOutBooks = new HashMap<>(MAX_BOOKS);
        this.previousCheckedOutBooks = new HashMap<>();
        this.inLibrary = false;
        this.currentFines = 0.0;
        this.totalFines = 0.0;
        this.payedFines = 0.0;
    }

    /**
     * Getter for the visitors name.
     * @return the first and last name combined
     */
    public String getName() {
        return this.firstName + " " + this.lastName;
    }

    /**
     * Getter for the username of a Visitor.
     * @return the visitors username
     */
    public String getUsername() {
        return this.username;
    }

    /**
     * Getter for the visitor's password.
     * @return the visitor's password
     */
    public String getPassword() {
        return this.password;
    }

    /**
     * Getter for the visitors address.
     * @return the visitors address
     */
    public String getAddress() {
        return this.address;
    }

    /**
     * Getter for the visitors phone number.
     * @return the visitors phone number
     */
    public PhoneNumber getPhoneNumber() {
        return this.phoneNumber;
    }

    /**
     * Getter for the visitors ID.
     * @return the visitors ID
     */
    public long getVisitorID() {
        return this.visitorID;
    }

    /**
     * Getter for the number of books the visitor has checked out.
```

```
     * @return the number of checked out books
     */
    public int getNumCheckedOut() {
        return this.checkedOutBooks.size();
    }

    /**
     * Getter for the checked out books
     * @return the checked out books
     */
    public HashMap<ISBN, Transaction> getCheckedOutBooks() {
        return this.checkedOutBooks;
    }

    /**
     * Getter for previously checked out books
     * @return previously checked out books
     */
    public HashMap<ISBN, Transaction> getPreviousCheckedOutBooks() {
        return this.previousCheckedOutBooks;
    }

    /**
     * Determines if a visitor can check out a book.
     * @return true if the number of checked out books is less than the
max
     */
    public boolean canCheckOut() {
        return getNumCheckedOut() < MAX_BOOKS && !(this.totalFines +
this.currentFines > this.payedFines);
    }

    /**
     * Checks out a book for a visitor.
     * @param transaction: the transaction for the checked out book
     */
    public void checkOut(Transaction transaction) {
        if (canCheckOut()) {
            this.checkedOutBooks.put(transaction.getIsbn(), transaction);
        }
    }

    /**
     * Undoes the action of a visitor checking out a book
     * @param transaction: the transaction for the checked out book
     */
    public void undoCheckOut(Transaction transaction) {
        this.checkedOutBooks.remove(transaction.getIsbn());
    }

    /**
     * Returns a book for a visitor.
     * @param transaction: the transaction created when the book was
checked out
     */
    public void returnBook(Transaction transaction) {
        this.totalFines += transaction.getFine();
        this.previousCheckedOutBooks.put(transaction.getIsbn(),
transaction);
        this.checkedOutBooks.remove(transaction.getIsbn());
```

```
    }

    /**
     * Undoes the return of a book for a visitor.
     * @param transaction: the transaction created when the book was
checked out
     */
    public void undoReturnBook(Transaction transaction) {
        this.totalFines -= transaction.getFine();
        this.checkedOutBooks.put(transaction.getIsbn(), transaction);
        this.previousCheckedOutBooks.remove(transaction.getIsbn());
    }

    /**
     * Getter for the status of the visitor.
     * @return true if the visitor is in the library, false if not
     */
    public boolean getInLibrary() {
        return this.inLibrary;
    }

    /**
     * Changes the in library status of a visitor.
     * @param status: a boolean of the status of a visitor
     */
    public void switchInLibrary(boolean status) {
        this.inLibrary = status;
    }

    /**
     * Determines the fines the visitor owes.
     * @return the amount of fines due
     */
    public double getFines() {
        double fines = 0;
        for (ISBN l: this.checkedOutBooks.keySet()) {
            fines += this.checkedOutBooks.get(l).getFine();
        }
        this.currentFines = fines;
        return this.currentFines + this.totalFines - this.payedFines;
    }

    /**
     * Makes a payment to the library.
     * @param amount: the amount of fines to pay
     */
    public void payFines(double amount) {
        this.payedFines += amount;
    }

    /**
     * Getter for payed fines.
     * @return the amount of fines this visitor has payed
     */
    public double getPayedFines() {
        return this.payedFines;
    }

    /**
     * Creates a string for a visitor.
```

```
     * @return a string representation of a visitor
     */
    @Override
    public String toString(){
        return "Firstname: " + this.firstName + "\nLastname: " +
this.lastName + "\nAddress: " + this.address +
                "\nPhone number: " + this.phoneNumber + "\nVisitorID: " +
this.visitorID + "\nUsername: " +
                this.username + "\nPassword: " + this.password + "\nIn-
library: " + this.inLibrary + "\nAll fines: " +
                this.getFines() + "\n";
    }

    /**
     * Setter for the username and password when an account is created.
     * @param username: the account's username
     * @param password: the account's password
     */
    public void setCredentials(String username, String password) {
        this.username = username;
        this.password = password;
    }
}
```

```
##############
# Visit.java #
##############
```

```
package lbms.models;

import java.io.Serializable;
import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

/**
 * Class for a Visit Object, used in the library book management system.
 * @author Team B
 */
public class Visit implements Serializable {

    private Visitor visitor;
    private LocalDateTime dateTime;
    private LocalTime timeOfDeparture;
    private Duration duration;

    /**
     * Constructor for a Visit object.
     * @param visitor: the ID of the visitor who is at the library
     */
    public Visit(Visitor visitor) {
        this.visitor = visitor;
        this.dateTime = SystemDateTime.getInstance(null).getDateTime();
        this.timeOfDeparture = null;
        this.duration = null;
        this.visitor.switchInLibrary(true);
    }

    /**
```

```
     * Departs the visitor from the library.
     */
    public void depart() {
        this.timeOfDeparture =
SystemDateTime.getInstance(null).getTime();
        this.duration = Duration.between(this.dateTime.toLocalTime(),
this.timeOfDeparture);
        this.visitor.switchInLibrary(false);
    }

    /**
     * Reverses a departure.
     */
    public void unDepart() {
        this.timeOfDeparture = null;
        this.duration = null;
        this.visitor.switchInLibrary(true);
    }

    /**
     * Getter for the visitor
     * @return the visitor
     */
    public Visitor getVisitor() {
        return this.visitor;
    }

    /**
     * Getter for the visit date.
     * @return local date of the visit
     */
    public LocalDate getDate() {
        return this.dateTime.toLocalDate();
    }

    /**
     * Getter for the arrival time.
     * @return local time for the arrival time
     */
    public LocalTime getArrivalTime() {
        return this.dateTime.toLocalTime();
    }

    /**
     * Getter for the departure time.
     * @return local time for the departure time
     */
    public LocalTime getDepartureTime() {
        return this.timeOfDeparture;
    }

    /**
     * Getter for the visit duration.
     * @return the duration of the visit
     */
    public Duration getDuration() {
        return this.duration;
    }
}
```

```
####################
# LibraryState.java #
####################

package lbms.models;

/**
 * LibraryStatus class created to used the state pattern.
 * @author Team B
 */
public interface LibraryState {

    /**
     * Determines if the library is open.
     * @return true if the library is open, false if not
     */
    boolean isOpen();

}

####################
# PhoneNumber.java #
####################

package lbms.models;

import java.io.Serializable;

/**
 * PhoneNumber class used in the Library Book Management System.
 * @author Team B
 */
public class PhoneNumber implements Serializable {

    private static final long serialVersionUID = 1L;
    private int areaCode;
    private int exchangeCode;
    private int extension;

    /**
     * Constructor for a PhoneNumber.
     * @param areaCode: the area code
     * @param exchangeCode: the first three digits
     * @param extension: the last 4 digits
     */
    public PhoneNumber(int areaCode, int exchangeCode, int extension) {
        this.areaCode = areaCode;
        this.exchangeCode = exchangeCode;
        this.extension = extension;
    }

    /**
     * Alternate constructor for a phone number.
     * @param s: the string to be converted to a phone number
     * @throws IllegalArgumentException: when the string cannot be turned
into a phone number
     */
    public PhoneNumber(String s) throws IllegalArgumentException {
        if (isValid(s)) {
            this.areaCode = Integer.valueOf(s.substring(0, 3));
```

```
            this.exchangeCode = Integer.valueOf(s.substring(3, 6));
            this.extension = Integer.valueOf(s.substring(6, 10));
        } else {
            throw new IllegalArgumentException("Not a valid phone
number");
        }
    }

    /**
     * Creates a string of the phone number.
     * @return a string representation for the phone number
     */
    @Override
    public String toString() {
        return String.format("%03d%03d%03d", this.areaCode,
this.exchangeCode, this.extension);
    }

    /**
     * Determines if the phone number is a valid phone number or not.
     * @param number the string of the phone number
     * @return true if valid, false if not
     */
    private static boolean isValid(String number) {
        return (number.length() == 10 && number.matches("-?\\d+?"));
    }
}
```

```
##################
# OpenState.java #
##################
```

```
package lbms.models;

/**
 * OpenState class used for the state of the library when it is open.
 * @author Team B
 */
public class OpenState implements LibraryState {

    /**
     * Returns true because when the library is in this state it is
always open.
     * @return true
     */
    public boolean isOpen() {
        return true;
    }

}
```

```
#######################
# SystemDateTime.java #
#######################
```

```
package lbms.models;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
```

```
import java.time.format.DateTimeFormatter;

/**
 * Custom date time implementation for the Library Book Management
System.
 * @author Team B
 */
public class SystemDateTime extends Thread {

    /** Formats for the date time. */
    private final static DateTimeFormatter DATETIME_FORMAT =
DateTimeFormatter.ofPattern("yyyy/MM/dd, HH:mm:ss");
    public final static DateTimeFormatter DATE_FORMAT =
DateTimeFormatter.ofPattern("yyyy/MM/dd");
    public final static DateTimeFormatter TIME_FORMAT =
DateTimeFormatter.ofPattern("HH:mm:ss");

    private static SystemDateTime instance = null;
    private LocalDateTime time;
    private volatile boolean stop = false;

    /**
     * Runs the thread for the clock.
     */
    @Override
    public void run() {
        while (!this.stop) {
            this.time = this.time.plusSeconds(1);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.err.print("");
            }
        }
    }

    /**
     * Constructor for a SystemDateTime object.
     */
    private SystemDateTime() {
        this.time = LocalDateTime.now();
    }

    /**
     * Constructor for a SystemDateTime object after de-serialization.
     * @param time: the time from the previous startup
     */
    private SystemDateTime(LocalDateTime time) {
        this.time = time;
    }

    /**
     * Gets the instance of the system date time, or creates a new one.
     * @return the instance of the system date time
     */
    private static SystemDateTime getInstance() {
        if (instance == null) {
            instance = new SystemDateTime();
        }
        return instance;
```

```
    }

    /**
     * Gets the instance of the system date time, or creates a new one,
may set the time.
     * @param time: the previous system time
     * @return an instance of the SystemDateTime
     */
    public static SystemDateTime getInstance(LocalDateTime time) {
        if (time == null) {
            return getInstance();
        } else if (instance == null) {
            instance = new SystemDateTime(time);
        }
        return instance;
    }

    /**
     * Gets the time of the system.
     * @return a local time object of the time
     */
    public LocalTime getTime() {
        return this.time.toLocalTime();
    }

    /**
     * Gets the date of the system.
     * @return a local date object of the system date
     */
    public LocalDate getDate() {
        return this.time.toLocalDate();
    }

    /**
     * Gets the system date time.
     * @return a local date time object of the system
     */
    public LocalDateTime getDateTime() {
        return this.time;
    }

    /**
     * Creates a string of the system date time.
     * @return string representation of the system date time
     */
    public String toString() {
        return this.time.format(DATETIME_FORMAT);
    }

    /**
     * Advances the time by days.
     * @param days: the number of days to advance the time
     */
    public void plusDays(long days) {
        this.time = this.time.plusDays(days);
    }

    /**
     * Advances the time by hours.
     * @param hours: the number of hours to advance the time
```

```
     */
    public void plusHours(long hours) {
        this.time = this.time.plusHours(hours);
    }

    /**
     * Resets the time.
     */
    public void reset() {
        this.time = LocalDateTime.now();
    }

    /**
     * Stops the clock.
     */
    public void stopClock() {
        this.stop = true;
    }
}

#################
# Employee.java #
#################

package lbms.models;

import java.io.Serializable;

/**
 * Class for an Employee object, used in the library book management
system.
 * @author Team B
 */
public class Employee implements Serializable {

    private Visitor v;

    /**
     * Constructor for an Employee object.
     * @param v: the visitor account for the employee
     */
    public Employee(Visitor v) {
        this.v = v;
    }

    /**
     * Getter for the employee's name.
     * @return the first and last name concatenated
     */
    public String getName() {
        return this.v.getName();
    }

    /**
     * Getter for the employee's username.
     * @return the username
     */
    public String getUsername() {
        return this.v.getUsername();
    }
```

```
    /**
     * Getter for the employee's password.
     * @return the password
     */
    public String getPassword() {
        return this.v.getPassword();
    }

    /**
     * Getter for the visitor of the employee.
     * @return the visitor account
     */
    public Visitor getVisitor() {
        return this.v;
    }

    /**
     * Creates a string of the employee's data.
     * @return a string representation of this object
     */
    @Override
    public String toString() {
        return "Employee:\n" + this.v.toString();
    }
}
```

```
#######################
# SessionManager.java #
#######################
```

```
package lbms.views.GUI;

import javafx.fxml.FXMLLoader;
import javafx.scene.control.Tab;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;

import java.util.HashMap;

/**
 * SessionManager class for the LBMS.
 * @author Team B
 */
public class SessionManager {

    private Tab tab;
    private Long clientId;
    private StateController controller;
    private Long visitorID;
    private String user;

    /**
     * Constructor for a SessionManager object.
     * @param tab: the tab for the session
     */
    public SessionManager(Tab tab) {
        try {
            // parse response
```

```
            HashMap<String, String> response =
ParseResponseUtility.parseResponse(
                    new
ProxyCommandController().processRequest("connect;"));
            this.clientId = Long.parseLong(response.get("clientID"));
        } catch (Exception e) {
            System.out.println("Error connecting to server.");
            System.exit(1);
        }
        this.tab = tab;
        this.visitorID = null;
    }

    /**
     * Closes the session.
     */
    public void close(boolean arg) {
        new ProxyCommandController().processRequest(this.clientId +
",logout;");
        new ProxyCommandController().processRequest(this.clientId +
",disconnect;");

        if (arg) {
            this.tab.getTabPane().getTabs().remove(this.tab);
        }
    }

    /**
     * Displays the session.
     * @param file: the fxml resource
     * @param title: title for the tab
     */
    public void display(String file, String title) {
        load(file);
        setTitle(title);
    }

    /**
     * Displays the session.
     * @param file: the fxml resource
     * @param title: title for the tab
     * @param flag: flag whether to include visitor ID or not
     */
    public void display(String file, String title, boolean flag) {
        load(file);
        setTitle(title, flag);
    }

    /**
     * Loads a file for the GUI formatting.
     * @param file: the filename
     */
    private void load(String file) {
        try {
            FXMLLoader loader = new FXMLLoader();
            loader.setLocation(SessionManager.class.getResource("/fxml/"
+ file + ".fxml"));
            this.tab.setContent(loader.load());
            this.controller = loader.getController();
            this.controller.initManager(this);
```

```
        } catch (Exception e) {
            System.out.println(e);
            System.out.println("Error loading fxml");
            System.exit(1);
        }
    }

    /**
     * Sets the title of the tab.
     * @param title: the title to be set
     */
    private void setTitle(String title) {
        this.tab.setText("Visitor ID: " + this.visitorID + " - " +
title);
    }

    /**
     * Sets the title, includes the visitorID if arg is true
     * @param title: the title for the tab
     * @param arg: boolean for including the visitorID
     */
    private void setTitle(String title, boolean arg) {
        if (arg) {
            this.tab.setText(this.visitorID + " - " + title);
        } else {
            this.tab.setText(title);
        }
    }

    /**
     * Getter for the client ID.
     * @return the client ID of the session
     */
    public Long getClientId() {
        return this.clientId;
    }

    /**
     * Setter for the user of the session.
     * @param visitorID: the visitorID of visitor connected to the
session
     */
    public void setVisitor(Long visitorID) {
        this.visitorID = visitorID;
    }

    /**
     * Getter for the user of the session
     * @return the user
     */
    public Long getVisitor() {
        return this.visitorID;
    }

    /**
     * Setter for the user.
     * @param user: the user to be set
     */
    public void setUser(String user) {
        this.user = user;
```

```
    }

    /**
     * Getter for the user.
     * @return the user
     */
    public String getUser() {
        return this.user;
    }

    /**
     * Getter for the controller.
     * @return the controller
     */
    public StateController getController() {
        return this.controller;
    }
}
```

```
################
# GUIView.java #
################
```

```
package lbms.views.GUI;

import javafx.application.Application;
import javafx.application.Platform;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import lbms.controllers.guicontrollers.ClientController;
import lbms.views.View;

/**
 * GUIView class used for the graphical user interface of the LBMS.
 * @author Team B
 */
public class GUIView extends Application implements View {

    /**
     * Constructor for a GUIView.
     */
    public GUIView() {}

    /**
     * Launches the GUI window.
     */
    @Override
    public void run() {
        launch();
    }

    /**
     * Starts the interface setup.
     * @param primaryStage: the first stage
     */
    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Library Book Management System");
```

```
        Parent root = null;
        try {
            FXMLLoader loader = new FXMLLoader();

loader.setLocation(GUIView.class.getResource("/fxml/client.fxml"));
            root = loader.load();
        } catch (Exception e) {
            System.out.println("Error loading fxml file");
            System.exit(1);
        }

        final String os = System.getProperty ("os.name");
        if (os != null && os.startsWith ("Mac")) {
            primaryStage.setScene(new Scene(root, 1280, 800));
        } else {
            primaryStage.setScene(new Scene(root, 1280, 850));
        }

        primaryStage.show();
    }

    /**
     * Used to stop the GUIView.
     */
    @Override
    public void stop(){
        ClientController.stop();
        Platform.exit();
    }

}

####################
# ViewFactory.java #
####################

package lbms.views;

import lbms.LBMS;
import lbms.views.API.APIView;
import lbms.views.GUI.GUIView;

/**
 * Controller for the views package.
 * @author Team B
 */
public class ViewFactory {

    /**
     * Starts up the LBMS by creating the user interface type and running
it.
     * @param type: the specified type of interface to use
     */
    public static void start(LBMS.StartType type) {
        switch (type) {
            case API:
                APIView.getInstance().run();
                break;
            case GUI:
```

```
                new GUIView().run();
                break;
            default:
                new GUIView().run();
                break;
        }
    }
}

################
# APIView.java #
################

package lbms.views.API;

import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.views.View;

import java.util.Scanner;

/**
 * APIView class used for the API mode of the LBMS.
 * @author Team B
 */
public class APIView implements View {

    private static APIView instance = null;

    /**
     * Constructor for the APIView.
     */
    private APIView() {}

    /**
     * Gets the instance or creates a new one.
     * @return an instance of the APIView.
     */
    public static APIView getInstance() {
        if (instance == null) {
            instance = new APIView();
        }
        return instance;
    }

    /**
     * Runs the API mode of the LBMS.
     */
    public void run() {
        Scanner s = new Scanner(System.in);
        String input;

        do {
            System.out.print("> ");
            input = s.nextLine();
            System.out.println(new
ProxyCommandController().processRequest(input));
        } while (!input.matches("(?i)exit|quit"));

        s.close();
    }
```

```
}

#############
# View.java #
#############

package lbms.views;

/**
 * View interface for the library book management system views.
 * @author Team B
 */
public interface View {

    /**
     * Used to start the view.
     */
    void run();

}

##############################
# MainVisitorController.java #
##############################

package lbms.controllers.guicontrollers;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import
lbms.controllers.guicontrollers.SearchControllers.LibrarySearchController
;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * MainVisitorController class used for controlling visitors.
 * @author Team B
 */
public class MainVisitorController implements StateController {

    private final static String BEGIN_VISIT_ID = "begin-visit-button";
    private final static String END_VISIT_ID = "end-visit-button";

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private TabPane searchTabPane;
    @FXML private Tab searchByAuthor;
    @FXML private Tab searchByTitle;
```

```
    @FXML private Tab searchByISBN;
    @FXML private TextField searchTitleField;
    @FXML private TextField searchAuthorField;
    @FXML private TextField searchISBNField;
    @FXML private Button visitButton;
    @FXML private Text failedLabel;

    /**
     * Initializes the state in this class.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search();
                e.consume();
            }
        });

        this.searchByAuthor.setUserData("author");
        this.searchByTitle.setUserData("title");
        this.searchByISBN.setUserData("isbn");
    }

    /**
     * Initializes the manager for the controller.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(final SessionManager manager) {
        this.manager = manager;

        if (ProxyCommandController.inLibrary(manager.getClientId())) {
            this.visitButton.setText("End Visit");
            this.visitButton.setOnAction(e -> endVisit());
            this.visitButton.setId(END_VISIT_ID);
        } else {
            this.visitButton.setText("Begin Visit");
            this.visitButton.setOnAction(e -> beginVisit());
            this.visitButton.setId(BEGIN_VISIT_ID);
        }
    }

    /**
     * Method used for setting up the search bars in the GUI.
     */
    public void search() {
        String author = this.searchAuthorField.getText();
        String title = this.searchTitleField.getText();
        String isbn = this.searchISBNField.getText();
        String type =
this.searchTabPane.getSelectionModel().getSelectedItem().getUserData().to
String();

        this.manager.display("search_library", "Library Search");

((LibrarySearchController)this.manager.getController()).search(type,
title, author, isbn);
    }
```

```
    /**
     * Begins a visit for the visitor.
     */
    private void beginVisit() {
        String request = String.format("%s,arrive;",
this.manager.getClientId());
        String response = new
ProxyCommandController().processRequest(request);

        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        switch (responseObject.get("message")) {
            case "invalid-id":
                this.failedLabel.setText("Visitor does not exist.");
                break;
            case "library-closed":
                this.failedLabel.setText("Sorry the library is closed,
please try again later.");
                break;
            case "duplicate":
                this.failedLabel.setText("Visitor is already in the
library.");
            default:
                this.visitButton.setText("End Visit");
                this.visitButton.setOnAction(e -> endVisit());
                this.visitButton.setId(END_VISIT_ID);
                break;
        }
    }

    /**
     * Ends a visit for the visitor.
     */
    private void endVisit() {
        String request = String.format("%s,depart;",
this.manager.getClientId());
        String response = new
ProxyCommandController().processRequest(request);

        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        switch (responseObject.get("message")) {
            case "invalid-id":
                this.failedLabel.setText("Visitor is currently not in the
library.");
            default:
                this.visitButton.setText("Begin Visit");
                this.visitButton.setOnAction(e -> beginVisit());
                this.visitButton.setId(BEGIN_VISIT_ID);
                break;
        }
    }

    /**
     * Logs out a visitor / employee.
     */
    @FXML
    public void logout() {
```

```
        new
ProxyCommandController().processRequest(this.manager.getClientId() +
",logout;");
        this.manager.display("login", "Login", false);
    }
}

##############################
# MainEmployeeController.java #
##############################

package lbms.controllers.guicontrollers;

import javafx.fxml.FXML;
import javafx.scene.control.Tab;
import javafx.scene.control.TabPane;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ProxyCommandController;
import
lbms.controllers.guicontrollers.SearchControllers.LibrarySearchController
;
import
lbms.controllers.guicontrollers.SearchControllers.StoreSearchController;
import lbms.views.GUI.SessionManager;

/**
 * MainEmployeeController class for the GUI of the Library Book
Management System.
 * @author Team B
 */
public class MainEmployeeController implements StateController {

    private SessionManager manager;

    @FXML private TabPane storeSearchBox;
    @FXML private TabPane searchBox;
    @FXML private Tab searchByAuthor;
    @FXML private Tab searchByTitle;
    @FXML private Tab searchByISBN;
    @FXML private Tab storeByAuthor;
    @FXML private Tab storeByTitle;
    @FXML private Tab storeByISBN;
    @FXML private TextField storeTitleField;
    @FXML private TextField storeAuthorField;
    @FXML private TextField storeISBNField;
    @FXML private TextField searchTitleField;
    @FXML private TextField searchAuthorField;
    @FXML private TextField searchISBNField;
    @FXML private Text failedLabel;

    /**
     * Initializes the state of the instance of this class.
     */
    @FXML
    protected void initialize() {

this.storeSearchBox.addEventHandler(javafx.scene.input.KeyEvent.KEY_PRESS
ED, e -> {
```

```
            if (e.getCode() == KeyCode.ENTER) {
                searchStore();
                e.consume();
            }
        });


this.searchBox.addEventHandler(javafx.scene.input.KeyEvent.KEY_PRESSED, e
-> {
            if (e.getCode() == KeyCode.ENTER) {
                search();
                e.consume();
            }
        });

        this.searchByTitle.setUserData("title");
        this.searchByAuthor.setUserData("author");
        this.searchByISBN.setUserData("isbn");
        this.storeByTitle.setUserData("title");
        this.storeByAuthor.setUserData("author");
        this.storeByISBN.setUserData("isbn");
    }

    /**
     * Initializes the manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Used for setting up the search bars in the GUI.
     */
    @FXML
    public void search() {
        String author = this.searchAuthorField.getText();
        String title = this.searchTitleField.getText();
        String isbn = this.searchISBNField.getText();
        String type =
this.searchBox.getSelectionModel().getSelectedItem().getUserData().toStri
ng();

        this.manager.display("search_library", "Library Search");

((LibrarySearchController)this.manager.getController()).search(type,
title, author, isbn);
    }

    /**
     * Sets up the search bar for the store search.
     */
    @FXML
    public void searchStore() {
        String author = this.storeAuthorField.getText();
        String title = this.storeTitleField.getText();
        String isbn = this.storeISBNField.getText();
```

```
        String type =
this.storeSearchBox.getSelectionModel().getSelectedItem().getUserData().t
oString();

        this.manager.display("search_store", "Store Search");

((StoreSearchController)this.manager.getController()).search(type, title,
author, isbn);
    }

    /**
     * Method for beginning a visit as an employee.
     */
    @FXML
    public void beginVisit() {
        if (ProxyCommandController.isOpen()) {
            this.manager.display("begin_visit", "Begin Visit");
        } else {
            this.failedLabel.setText("Sorry the library is closed, please
try again later.");
        }
    }

    /**
     * Ends a visit for an employee.
     */
    @FXML
    public void endVisit() {
        if (ProxyCommandController.isOpen()) {
            this.manager.display("end_visit", "End Visit");
        } else {
            this.failedLabel.setText("Sorry the library is closed, please
try again later.");
        }
    }

    /**
     * Returns a book.
     */
    @FXML
    public void returnBook() {
        this.manager.display("return", "Return Book");
    }

    /**
     * Method for an employee registering a visitor.
     */
    @FXML
    public void register() {
        this.manager.display("register", "Register Visitor");
    }

    /**
     * Method for creating an account as an employee.
     */
    @FXML
    public void create() {
        this.manager.display("create", "Create Account");
    }
```

```
    /**
     * Method for going to the system settings as an employee.
     */
    @FXML
    public void settings() {
        this.manager.display("settings", "System Settings");
    }

    /**
     * Logs out an employee.
     */
    @FXML
    public void logout() {
        new
ProxyCommandController().processRequest(this.manager.getClientId() +
",logout;");
        this.manager.display("login", "Login", false);
    }
}

################################
# BorrowSuccessController.java #
################################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

import java.util.HashMap;

/**
 * BorrowSuccessController class for the Library Book Management System.
 * @author Team B
 */
public class BorrowSuccessController {

    private HashMap<String, String> book;
    private String visitor, dueDate;

    @FXML private AnchorPane root;
    @FXML private Text title, date;

    /**
     * Initializes the state for this class.
     */
    @FXML
    protected void initialize() {

this.root.addEventHandler(javafx.scene.input.KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                close();
                e.consume();
            }
        });
    }
```

```
    /**
     * Loads any necessary data.
     * @param book: hash map of date for the book to be loaded
     * @param visitor: the visitor borrowing books
     * @param dueDate: the date the books will be due
     */
    void load(HashMap<String, String> book, String visitor, String
dueDate) {
        this.book = book;
        this.visitor = visitor;
        this.dueDate = dueDate;
        display();
    }

    /**
     * Displays the book.
     */
    private void display() {
        this.title.setText("Visitor " + this.visitor + " has borrowed " +
this.book.get("title"));
        this.date.setText("Due Date: " + this.dueDate);
    }

    /**
     * Closes the stage.
     */
    @FXML
    public void close() {
        Stage stage = (Stage)this.title.getScene().getWindow();
        stage.close();
    }
}
##############################
# StoreSearchController.java #
##############################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.RadioButton;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * StoreSearchController class for the Library Book Management System.
 * @author Team B
 */
public class StoreSearchController implements StateController {
```

```
    private SessionManager manager;

    @FXML private VBox results;
    @FXML private Text noResultsLabel;
    @FXML private TextField titleField, authorField, isbnField;
    @FXML private RadioButton localStore, googleStore;

    /**
     * Initializes the data for this class.
     */
    @FXML
    protected void initialize() {
        ToggleGroup group = new ToggleGroup();

        this.localStore.setToggleGroup(group);
        this.localStore.setUserData("local");
        this.localStore.setOnAction(e -> service(this.localStore));

        this.googleStore.setToggleGroup(group);
        this.googleStore.setUserData("google");
        this.googleStore.setOnAction(e -> service(this.googleStore));

        this.titleField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search("title", this.titleField.getText(), "", "");
                e.consume();
            }
        });
        this.authorField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search("author", "", this.authorField.getText(), "");
                e.consume();
            }
        });
        this.isbnField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search("isbn", "", "", this.isbnField.getText());
                e.consume();
            }
        });
    }

    /**
     * Setter for the session manager.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;

        if
(ProxyCommandController.getStore(manager.getClientId()).equals("local"))
{
            this.localStore.setSelected(true);
        } else {
            this.googleStore.setSelected(true);
        }
    }
```

```
    /**
     * Creates a string request, processes it, and displays the result.
     * @param type: the type of search
     * @param title: the title for the search
     * @param author: the author for the search
     * @param isbn: the isbn for the search
     */
    public void search(String type, String title, String author, String
isbn) {
        String request;
        switch (type) {
            case "author":
                request = String.format("%s,search,*,{%s};",
this.manager.getClientId(), author);
                break;
            case "title":
                request = String.format("%s,search,%s,*;",
this.manager.getClientId(), title);
                break;
            case "isbn":
                request = String.format("%s,search,*,*,%s;",
this.manager.getClientId(), isbn);
                break;
            default:
                request = null;
                break;
        }

        String response = new
ProxyCommandController().processRequest(request);
        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        display(responseObject);
    }

    /**
     * Displays the data in the GUI.
     * @param response: the response hash map of data
     */
    private void display(HashMap<String, String> response) {
        this.titleField.clear();
        this.authorField.clear();
        this.isbnField.clear();
        this.results.getChildren().clear();
        this.noResultsLabel.setText("");

        if (Integer.parseInt(response.get("numberOfBooks")) == 0) {
            this.noResultsLabel.setText("No Books Found");
        } else {
            ArrayList<HashMap<String, String>> books =
ParseResponseUtility.parseBooks(response.get("books"));
            for (HashMap<String, String> book: books) {
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/book.fxml"));
                    this.results.getChildren().add(loader.load());

((SearchResultController)loader.getController()).load(this.manager, book,
false);
```

```
                } catch (Exception e) {
                    System.out.println("Error loading book.");
                }
            }
        }
    }

    /**
     * Controls the button for the search service.
     * @param button: the button for the search service
     */
    private void service(RadioButton button) {
        String request = this.manager.getClientId() + ",service," +
button.getUserData() + ";";
        String response = new
ProxyCommandController().processRequest(request);

        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        if (!responseObject.get("message").equals("success")) {
            System.out.println("error occurred");
        }
    }

    /**
     * Searches by title.
     */
    @FXML
    public void titleSearch() {
        search("title", this.titleField.getText(), "", "");
    }

    /**
     * Searches based on the author.
     */
    @FXML
    public void authorSearch() {
        search("author", "", this.authorField.getText(), "");
    }

    /**
     * Searches based on ISBN.
     */
    @FXML
    public void isbnSearch() {
        search("isbn", "", "", this.isbnField.getText());
    }

    /**
     * Goes to the home page for the visitor / employee.
     */
    @FXML
    public void home() {
        if
(ProxyCommandController.isEmployee(this.manager.getClientId())) {
            this.manager.display("main_employee",
this.manager.getUser());
        } else {
            this.manager.display("main_visitor", this.manager.getUser());
        }
```

```
    }
}

##############################
# SearchResultController.java #
##############################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Hyperlink;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * SearchResultController class for the Library Book Management System.
 * @author Team B
 */
public class SearchResultController {

    private SessionManager manager;
    private boolean state;
    private HashMap<String, String> book;

    @FXML private Hyperlink isbn;
    @FXML private Text title;
    @FXML private Text author;
    @FXML private Text publishDate;
    @FXML private Text publisher;
    @FXML private Text pageCount;

    /**
     * Loads the data into this class.
     * @param manager: the session manager for the class
     * @param book: the book data
     * @param arg: the state for being borrowed of purchased
     */
    void load(SessionManager manager, HashMap<String, String> book,
boolean arg) {
        this.manager = manager;
        this.state = arg;
        this.book = book;
        populate();
    }

    /**
     * Populates the GUI parts with the data.
     */
    private void populate() {
         this.isbn.setText(this.book.get("isbn"));
         this.title.setText(this.book.get("title"));
         this.author.setText(this.book.get("authors"));
         this.pageCount.setText(this.book.get("pageCount"));
         this.publishDate.setText(this.book.get("publishDate"));
```

```
         this.publisher.setText(this.book.get("publisher"));
    }

    /**
     * Selects the fxml file.
     */
    @FXML public void select() {
        Parent root;
        try {
            FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/book_info.fxml
"));
            root = loader.load();
            Stage stage = new Stage();
            stage.setTitle(this.manager.getVisitor() + " - " +
this.book.get("title"));
            stage.setScene(new Scene(root, 750, 500));

            ((BookInfoController)loader.getController()).load(stage,
this.manager, this.book, this.state);

            stage.show();
        } catch (Exception e) {
            System.out.println("Error loading FXML file.");
            System.exit(1);
        }
    }
}
```

```
##################################
# PurchaseSuccessController.java #
##################################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.text.Text;
import javafx.stage.Stage;

import java.util.HashMap;

/**
 * PurchaseSuccessController class for the Library Book Management
System.
 * @author Team B
 */
public class PurchaseSuccessController {

    private HashMap<String, String> book;

    @FXML private Text title, quantity;

    /**
     * Initializes the data for this instance of the class.
     */
    @FXML
    protected void initialize() {
```

```
this.title.getParent().addEventHandler(javafx.scene.input.KeyEvent.KEY_PR
ESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                close();
                e.consume();
            }
        });
    }

    /**
     * Loads the book to be displayed.
     * @param book: hash map of the information for a book
     */
    void load(HashMap<String, String> book) {
        this.book = book;
        display();
    }

    /**
     * Displays the data for a purchase.
     */
    private void display() {
        this.title.setText("Successfully purchased " +
this.book.get("title"));

        if (Integer.parseInt(this.book.get("quantity")) == 1) {
            this.quantity.setText("1 copy purchased");
        } else {
            this.quantity.setText(this.book.get("quantity") +  " copies
purchased");
        }
    }

    /**
     * Closes the stage.
     */
    @FXML
    public void close() {
        Stage stage = (Stage)this.title.getScene().getWindow();
        stage.close();
    }
}

############################
# BookInfoController.java #
############################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import lbms.controllers.commandproxy.ParseResponseUtility;
```

```
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.views.GUI.SessionManager;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * BookInfoController class for the Library Book Management System.
 * @author Team B
 */
public class BookInfoController {

    private Stage stage;
    private SessionManager manager;
    private HashMap<String, String> book;
    private boolean state;

    @FXML private Button actionButton;
    @FXML private Text title, author, isbn, publisher, publishDate,
pageCount, quantity;
    @FXML private Text quantityLabel, failedLabel, inputLabel;
    @FXML private TextField input;

    /**
     * Initializes the state for this class.
     */
    @FXML
    protected void initialize() {

this.actionButton.getParent().addEventHandler(javafx.scene.input.KeyEvent
.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                this.actionButton.fire();
                e.consume();
            }
        });
    }

    /**
     * Loads the book information for this page.
     * @param stage: the stage for the GUI
     * @param manager: the session manager
     * @param book: the hash map of books
     * @param state: the state boolean
     */
    void load(Stage stage, SessionManager manager, HashMap<String,
String> book, boolean state) {
        this.stage = stage;
        this.manager = manager;
        this.book = book;
        this.state = state;
        display();
    }

    /**
     * Displays the data.
     */
    private void display() {
        this.title.setText(this.book.get("title"));
        this.author.setText(this.book.get("authors"));
```

```
        this.isbn.setText(this.book.get("isbn"));
        this.publishDate.setText(this.book.get("publishDate"));
        this.publisher.setText(this.book.get("publisher"));
        this.pageCount.setText(this.book.get("pageCount"));

        if (this.state) {
            boolean status =
ProxyCommandController.isEmployee(this.manager.getClientId());

            this.quantityLabel.setText("Quantity");
            this.quantity.setText(this.book.get("quantity"));
            this.actionButton.setText("Borrow");
            this.actionButton.setOnAction(e -> borrow());
            this.inputLabel.setText("Visitor ID");

            if (!status) {
                this.input.setText(this.manager.getVisitor().toString());
                this.input.setDisable(true);
            }

        } else {
            this.quantityLabel.setText("");
            this.actionButton.setText("Purchase");
            this.actionButton.setOnAction(e -> purchase());
            this.inputLabel.setText("Quantity");
        }
    }

    /**
     * Borrows the book being viewed.
     */
    private void borrow() {
        if (this.input.getText().isEmpty()) {
            this.failedLabel.setText("Please enter a visitor ID.");
        } else {
            String request = String.format("%s,borrow,{%s},%s;",
this.manager.getClientId(), this.book.get("id"),
                    this.input.getText());

            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("success")) {
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/borrow_success
.fxml"));
                    Parent root = loader.load();

((BorrowSuccessController)loader.getController()).load(this.book,
this.input.getText(),
                            responseObject.get("dueDate"));
                    this.stage.setScene(new Scene(root, 750, 500));
                } catch (Exception e) {
                    e.printStackTrace();
                }
```

```
            } else if (responseObject.get("message").equals("no-more-
copies")){
                this.failedLabel.setText("There are no more copies of
this book. Please try again later.");
            } else if (responseObject.get("message").equals("book-limit-
exceeded")){
                this.failedLabel.setText("This visitor has exceeded their
book limit. Please return a book then try " +
                        "again");
            } else if (responseObject.get("message").equals("outstanding-
fine")){
                this.failedLabel.setText("This visitor has an outstanding
fine. Please pay fine then try again.");
            } else if (responseObject.get("message").equals("library-
closed")) {
                this.failedLabel.setText("Sorry the library is closed.
Please try again later.");
            } else {
                this.failedLabel.setText("Visitor does not exist. Please
try again.");
            }
        }
    }

    /**
     * Purchases the book for the library.
     */
    private void purchase() {
        if (this.input.getText().isEmpty()) {
            this.failedLabel.setText("Please enter a quantity.");
        } else {
            String request = String.format("%s,buy,%s,%s;",
this.manager.getClientId(), this.input.getText(),
                    this.book.get("id"));

            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("success")) {
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/purchase_succe
ss.fxml"));
                    Parent root = loader.load();
                    ArrayList<HashMap<String, String>> books =
ParseResponseUtility.parseBooks(responseObject
                            .get("books"));

((PurchaseSuccessController)loader.getController()).load(books.get(0));

                    this.stage.setScene(new Scene(root, 750, 500));
                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                this.failedLabel.setText("Purchase failure. Please try
again.");
```

```
            }
        }
    }
}
################################
# LibrarySearchController.java #
################################

package lbms.controllers.guicontrollers.SearchControllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * LibrarySearchController class for the library book management system.
 * @author Team B
 */
public class LibrarySearchController implements StateController {

    private SessionManager manager;

    @FXML private VBox results;
    @FXML private Text noResultsLabel;
    @FXML private TextField titleField;
    @FXML private TextField authorField;
    @FXML private TextField isbnField;

    /**
     * Initializes the data for this class.
     */
    @FXML
    protected void initialize() {
        this.titleField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search("title", this.titleField.getText(), "", "");
                e.consume();
            }
        });

        this.authorField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                search("author", "", this.authorField.getText(), "");
                e.consume();
            }
        });

        this.isbnField.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
```

```
                search("isbn", "", "", this.isbnField.getText());
                e.consume();
            }
        });
    }

    /**
     * Setter for the session manager.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Searches the library.
     * @param type: the type of search
     * @param title: the title being searched
     * @param author: the author for the search
     * @param isbn: the isbn of the search
     */
    public void search(String type, String title, String author, String
isbn) {
        String request;
        switch (type) {
            case "author":
                request = String.format("%s,info,*,{%s};",
this.manager.getClientId(), author);
                break;
            case "title":
                request = String.format("%s,info,%s,*;",
this.manager.getClientId(), title);
                break;
            case "isbn":
                request = String.format("%s,info,*,*,%s;",
this.manager.getClientId(), isbn);
                break;
            default:
                request = null;
                break;
        }

        String response = new
ProxyCommandController().processRequest(request);
        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        display(responseObject);
    }

    /**
     * Displays the library search data.
     * @param response: the hash map responses
     */
    private void display(HashMap<String, String> response) {
        this.titleField.clear();
        this.authorField.clear();
        this.isbnField.clear();
        this.results.getChildren().clear();
        this.noResultsLabel.setText("");
```

```
        if (Integer.parseInt(response.get("numberOfBooks")) == 0) {
            this.noResultsLabel.setText("No Books Found");
        } else {
            ArrayList<HashMap<String, String>> books =
ParseResponseUtility.parseBooks(response.get("books"));

            for (HashMap<String, String> book: books) {
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/book.fxml"));
                    this.results.getChildren().add(loader.load());

((SearchResultController)loader.getController()).load(this.manager, book,
true);
                } catch (Exception e) {
                    System.out.println("Error loading book.");
                }
            }
        }
    }

    /**
     * The title for the search.
     */
    @FXML
    public void titleSearch() {
        search("title", this.titleField.getText(), "", "");
    }

    /**
     * The author for the search.
     */
    @FXML
    public void authorSearch() {
        search("author", "", this.authorField.getText(), "");
    }

    /**
     * The isbn for the search.
     */
    @FXML
    public void isbnSearch() {
        search("isbn", "", "", this.isbnField.getText());
    }

    /**
     * Goes back to the home page for that person.
     */
    @FXML
    public void home() {
        if
(ProxyCommandController.isEmployee(this.manager.getClientId())) {
            this.manager.display("main_employee",
this.manager.getUser());
        } else {
            this.manager.display("main_visitor", this.manager.getUser());
        }
    }
```

```
}

########################
# StateController.java #
########################

package lbms.controllers.guicontrollers;

import lbms.views.GUI.SessionManager;

/**
 * StateController interface for the GUI.
 * @author Team B
 */
public interface StateController {

    /**
     * Initialized the manager for the controller.
     * @param manager: the session manager to be set
     */
    void initManager(SessionManager manager);

}

#########################
# CreateController.java #
#########################

package lbms.controllers.guicontrollers.RegisterControllers;

import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * CreateController class for the Library Book Management System.
 * @author Team B
 */
public class CreateController implements StateController {

    private SessionManager manager;
    private ToggleGroup group;

    @FXML private AnchorPane root;
    @FXML private TextField visitorField;
    @FXML private TextField usernameField;
    @FXML private PasswordField passwordField;
    @FXML private PasswordField confirmField;
    @FXML private RadioButton visitorButton;
    @FXML private RadioButton employeeButton;
    @FXML private Text visitorFail;
```

```
    @FXML private Text usernameFail;
    @FXML private Text passwordFail;
    @FXML private Text confirmFail;
    @FXML private Text roleFail;
    @FXML private Text failedLabel;
    @FXML private Button createButton;

    /**
     * Initializes the controller.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                this.createButton.fire();
                e.consume();
            }
        });

        this.group = new ToggleGroup();
        this.visitorButton.setToggleGroup(this.group);
        this.visitorButton.setUserData("visitor");
        this.employeeButton.setToggleGroup(this.group);
        this.employeeButton.setUserData("employee");
        this.visitorButton.setSelected(true);
    }

    /**
     * Sets the session manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Setter for the text of the visitor.
     * @param visitorID: the id of the visitor
     */
    public void setVisitor(String visitorID) {
        this.visitorField.setText(visitorID);
    }

    /**
     * Submit function used when the submit button is pressed.
     */
    @FXML
    private void create() {
        clearError();
        boolean completed = true;

        String visitor = this.visitorField.getText();
        String username = this.usernameField.getText();
        String password = this.passwordField.getText();
        String confirm = this.confirmField.getText();
        String role =
this.group.getSelectedToggle().getUserData().toString();

        if (visitor.isEmpty()) {
```

```
            completed = false;
            this.visitorFail.setText("*");
        }
        if (username.isEmpty()) {
            completed = false;
            this.usernameFail.setText("*");
        }
        if (password.isEmpty()) {
            completed = false;
            this.passwordFail.setText("*");
        }
        if (confirm.isEmpty()) {
            completed = false;
            this.confirmFail.setText("*");
        }
        if (role.isEmpty()) {
            completed = false;
            this.roleFail.setText("*");
        }

        if (!completed) {
            this.failedLabel.setText("* Please enter missing fields.");
        } else if (!password.equals(confirm)) {
            this.failedLabel.setText("Passwords do not match.\nPlease
enter your password again.");
        } else {
            boolean valid;
            try {
                String request = String.format("%s,create,%s,%s,%s,%s;",
                        this.manager.getClientId(), username, password,
role, visitor);

                String response = new
ProxyCommandController().processRequest(request);

                HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
                switch (responseObject.get("message")) {
                    case "duplicate-username":
                        this.failedLabel.setText("Username is taken.
Please try a new username.");
                        valid = false;
                        break;
                    case "duplicate-visitor":
                        this.failedLabel.setText("Visitor already has an
account. Please try logging in.");
                        valid = false;
                        break;
                    case "invalid-visitor":
                        this.failedLabel.setText("No visitor exists.
Please register as a visitor first.");
                        valid = false;
                        break;
                    default:
                        valid = true;
                        break;
                }
            } catch (Exception e) {
                valid = false;
            }
```

```
            if (valid) {
                this.manager.display("created_account", "Account
Created");

((AccountCreatedController)this.manager.getController()).setUsername(user
name);
            }
        }
    }

    /**
     * Tells the manager to cancel the last action.
     */
    @FXML
    public void cancel() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Clears all the fields by setting them to empty strings.
     */
    private void clearError() {
        this.visitorFail.setText("");
        this.usernameFail.setText("");
        this.passwordFail.setText("");
        this.confirmFail.setText("");
        this.roleFail.setText("");
    }
}

############################
# RegisterController.java #
############################

package lbms.controllers.guicontrollers.RegisterControllers;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;


/**
 * RegisterController class for the Library Book Management System.
 * @author Team B
 */
public class RegisterController implements StateController {

    private SessionManager manager;
    private String visitorId = "";
```

```
@FXML private AnchorPane root;
@FXML private Button registerButton;
@FXML private TextField firstNameField;
@FXML private TextField lastNameField;
@FXML private TextField addressField;
@FXML private TextField phoneNumberField;
@FXML private Text firstNameFail;
@FXML private Text lastNameFail;
@FXML private Text addressFail;
@FXML private Text phoneNumberFail;
@FXML private Text failedLabel;

/**
 * Initializes the manager.
 * @param manager: the session manager to be set
 */
public void initManager(final SessionManager manager) {
    this.manager = manager;
}

/**
 * Initializes the state of this class.
 */
@FXML
protected void initialize() {
    this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
        if (e.getCode() == KeyCode.ENTER) {
            this.registerButton.fire();
            e.consume();
        }
    });
}

/**
 * Register method registers a visitor.
 */
@FXML
private void register() {
    clearError();
    boolean completed = true;

    String firstName = this.firstNameField.getText();
    String lastName = this.lastNameField.getText();
    String address = this.addressField.getText();
    String phoneNumber = this.phoneNumberField.getText();

    if (firstName.isEmpty()) {
        completed = false;
        this.firstNameFail.setText("*");
    }
    if (lastName.isEmpty()) {
        completed = false;
        this.lastNameFail.setText("*");
    }
    if (address.isEmpty()) {
        completed = false;
        this.addressFail.setText("*");
    }
    if (phoneNumber.isEmpty()) {
```

```
            completed = false;
            this.phoneNumberFail.setText("*");
        }

        if (completed) {
            boolean valid = true;

            if (this.visitorId.isEmpty()) {
                try {
                    String request =
String.format("%s,register,%s,%s,%s,%s;",
                            this.manager.getClientId(), firstName,
lastName, address, phoneNumber);

                    String response = new
ProxyCommandController().processRequest(request);

                    HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
                    if
(responseObject.get("message").equals("duplicate")) {
                        valid = false;
                        this.failedLabel.setText("This visitor already
exists.\nPlease try again "
                                + "or register with visitor ID.");
                    } else if
(responseObject.get("message").equals("missing-parameters")) {
                        valid = false;
                        this.failedLabel.setText("One of the fields is
invalid. Please try again.");
                    } else {
                        this.visitorId = responseObject.get("visitorID");
                    }
                } catch (Exception e) {
                    valid = false;
                }
            }

            if (valid) {
                this.manager.display("registered_visitor", "Visitor
Registered");

((VisitorRegisteredController)this.manager.getController()).setVisitor(th
is.visitorId);
            }
        } else {
            this.failedLabel.setText("* Please enter missing fields.");
        }
    }

    /**
     * Tells the session manager to cancel the last action.
     */
    @FXML
    public void cancel() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Clears the error with the form.
```

```
     */
    private void clearError() {
        this.firstNameFail.setText("");
        this.lastNameFail.setText("");
        this.addressFail.setText("");
        this.phoneNumberFail.setText("");
    }
}

###################################
# VisitorRegisteredController.java #
###################################

package lbms.controllers.guicontrollers.RegisterControllers;

import javafx.fxml.FXML;
import javafx.scene.text.Text;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

/**
 * VisitorRegisteredController class for the GUI of the Library Book
Management System.
 * @author Team B
 */
public class VisitorRegisteredController implements StateController {

    private SessionManager manager;

    @FXML private Text visitorID;

    /**
     * Initializes the manager for the session manager.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Displays the create account form from the session manager.
     */
    @FXML
    public void yes() {
        this.manager.display("create", "Create Account");

((CreateController)this.manager.getController()).setVisitor(this.visitorI
D.getText());
    }

    /**
     * Tells the session manager that a computer account will not be
created.
     */
    @FXML
    public void no() {
        this.manager.display("main_employee", this.manager.getUser());
    }
```

```
    /**
     * Setter for the visitor ID.
     * @param visitorId: the visitor's ID
     */
    public void setVisitor(String visitorId) {
        this.visitorID.setText(visitorId);
    }
}

################################
# AccountCreatedController.java #
################################

package lbms.controllers.guicontrollers.RegisterControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

/**
 * AccountCreatedController for the GUI part of the Library Book
Management System.
 * @author Team B
 */
public class AccountCreatedController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private Text username;

    /**
     * Initializes the controller.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                home();
                e.consume();
            }
        });
    }

    /**
     * Initializes the manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Tells the manager to display the home stage.
     */
```

```
    @FXML
    public void home() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Sets the username for this class.
     * @param username: the username to be set
     */
    public void setUsername(String username) {
        this.username.setText(username);
    }
}
```

```
########################
# SystemController.java #
########################
```

```
package lbms.controllers.guicontrollers;

import javafx.fxml.FXML;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * SystemController class for the Library Book Management System.
 * @author Team B
 */
public class SystemController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private Text label;
    @FXML private TextArea output;
    @FXML private TextField input;
    @FXML private Text inputFail;
    @FXML private VBox timeBox;
    @FXML private TextField daysField;
    @FXML private TextField hoursField;
    @FXML private VBox reportBox;
    @FXML private TextField reportField;
    @FXML private TextArea reportOutput;

    /**
     * Initializes the state of this class.
     */
    @FXML
    protected void initialize() {
```

```
        this.output.setEditable(false);
        this.output.textProperty().addListener(c -> {
            this.output.setScrollTop(Double.MAX_VALUE);
        });

        this.reportOutput.setEditable(false);
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                command();
                e.consume();
            }
        });

        this.timeBox.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                advance();
                e.consume();
            }
        });

        this.reportBox.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                report();
                e.consume();
            }
        });
    }

    /**
     * Initializes the manager for this instance of the class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Displays the home stage.
     */
    @FXML
    public void home() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Advances the time for the system.
     */
    @FXML
    public void advance() {
        this.inputFail.setText("");
        this.label.setText("");
        this.label.setFill(Color.FIREBRICK);

        String days = this.daysField.getText();
        String hours = this.hoursField.getText();

        if (days.isEmpty() && hours.isEmpty()) {
            this.label.setText("Please enter days or hours to advance.");
        } else {
```

```
            if (days.isEmpty()) {
                days = "0";
            }
            if (hours.isEmpty()) {
                hours = "0";
            }

            String request = String.format("%s,advance,%s,%s;",
this.manager.getClientId(), days, hours);
            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            switch (responseObject.get("message")) {
                case "invalid-number-of-hours":
                    this.label.setText("Can not advance " + hours + "
hours. Please try again.");
                    break;
                case "invalid-number-of-days":
                    this.label.setText("Can not advance " + days + "
days. Please try again.");
                    break;
                default:
                    this.label.setText("Success");
                    this.label.setFill(Color.GREEN);
                    break;
            }

            this.daysField.setText("");
            this.hoursField.setText("");
        }
    }

    /**
     * Used for the system report viewing as an employee.
     */
    @FXML
    public void report() {
        this.inputFail.setText("");
        this.label.setText("");

        String request;
        String days = this.reportField.getText();

        if (days.isEmpty()) {
            request = this.manager.getClientId() + ",report;";
        } else {
            request = String.format("%s,report,%s;",
this.manager.getClientId(), days);
        }

        String response = new
ProxyCommandController().processRequest(request);

        HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
        if (responseObject.get("message").equals("success")) {
            this.reportOutput.setText("Date: " +
responseObject.get("date") + responseObject.get("report"));
```

```
        } else {
            this.reportOutput.setText("An error occurred.\nPlease try
again later.");
        }
    }

    /**
     * Used to enter commands directly into the Library Book Management
System.
     */
    @FXML
    public void command() {
        this.label.setText("");
        String request = this.input.getText();

        if (request.isEmpty()) {
            this.inputFail.setText("No input. Please enter a search.");
        } else {
            this.inputFail.setText("");
            String response = new
ProxyCommandController().processRequest(this.manager.getClientId() + ","
                    + request);

            try {
                HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);

                if (Long.parseLong(responseObject.get("clientID")) ==
this.manager.getClientId() &&
                        responseObject.get("command").equals("logout")) {
                    this.manager.display("login", "Login", false);
                } else if (Long.parseLong(responseObject.get("clientID"))
== this.manager.getClientId() &&

responseObject.get("command").equals("disconnect")) {
                    this.manager.close(true);
                } else {
                    throw new Exception();
                }
            } catch (Exception e) {
                this.output.appendText(request + "\n");
                this.output.appendText(response + "\n");
                this.input.clear();
            }
        }
    }
}

#########################
# ClientController.java #
#########################

package lbms.controllers.guicontrollers;

import javafx.application.Platform;
import javafx.event.ActionEvent;
import javafx.event.Event;
import javafx.fxml.FXML;
import javafx.scene.control.*;
import javafx.scene.input.KeyCode;
```

```
import javafx.scene.input.KeyCodeCombination;
import javafx.scene.input.KeyCombination;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.CommandController;
import lbms.views.GUI.SessionManager;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

/**
 * ClientController class for the Library Book Management System.
 * @author Team B
 */
public class ClientController {

    private static Boolean stop = false;

    @FXML private BorderPane root;
    @FXML private TabPane tabs;
    @FXML private Text clockText;

    /**
     * Initializes the state for this instance of the class.
     */
    @FXML
    protected void initialize() {
        createMenuBar();
        this.clockText.setFont(Font.font(null,FontWeight.BOLD, 13));

        Runnable task = () -> {
            while (!stop) {
                LocalDateTime date =
CommandController.getSystemDateTime();

this.clockText.setText(date.format(DateTimeFormatter.ofPattern("HH:mm
MM/dd/yyyy")));
            }
        };
        new Thread(task).start();

        Tab addTab = new Tab();
        addTab.setId("addTab");
        addTab.setGraphic(new Button());

        Button btn = ((Button)addTab.getGraphic());
        btn.setOnMouseClicked(event -> addTab());
        btn.setBorder(null);
        btn.setMinSize(31, 26);
        btn.setMaxSize(31,26);
        btn.setId("addTabButton");
        btn.setText("+");
        btn.setTextFill(Color.BLACK);

        addTab.setClosable(false);
        this.tabs.getTabs().add(addTab);
        addTab();
```

```
    }

    /**
     * Adds a tab to the window.
     */
    @FXML
    private void addTab() {
        int num = this.tabs.getTabs().size();
        Tab tab = new Tab("Login");

        SessionManager manager = new SessionManager(tab);
        manager.display("login", "Login", false);

        tab.setOnCloseRequest((Event event) -> { manager.close(false);
});
        this.tabs.getTabs().add(num - 1, tab);
        this.tabs.getSelectionModel().select(tab);
    }

    /**
     * Stops the GUI.
     */
    public static void stop() {
        stop = true;
    }

    /**
     * Creates the menu.
     */
    private void createMenuBar() {
        // Create Menu Bar
        MenuBar menuBar = new MenuBar();

menuBar.getStylesheets().add(getClass().getResource("/css/client.css").to
ExternalForm());

        // Add Menu
        KeyCombination.Modifier key;
        final String os = System.getProperty ("os.name");
        if (os != null && os.startsWith ("Mac")) {
            menuBar.useSystemMenuBarProperty().set(true);
            this.root.getChildren().add(menuBar);
            key = KeyCombination.META_DOWN;
        } else {
            this.root.setTop(menuBar);
            key = KeyCombination.CONTROL_DOWN;
        }

        // File
        Menu fileMenu = new Menu("File");
        MenuItem newTab = new MenuItem("New Tab");
        newTab.setOnAction((ActionEvent event) -> { addTab(); });
        newTab.setAccelerator(new KeyCodeCombination(KeyCode.T, key));
        MenuItem closeTab = new MenuItem("Close Tab");

        closeTab.setOnAction((ActionEvent event) -> {
            if
(this.tabs.getSelectionModel().getSelectedItem().isClosable()) {
```

```
this.tabs.getTabs().remove(this.tabs.getSelectionModel().getSelectedItem(
));
            }
        });

        closeTab.setAccelerator(new KeyCodeCombination(KeyCode.W, key));


        MenuItem closeWindow = new MenuItem("Close Window");
        closeWindow.setOnAction((ActionEvent event) -> { Platform.exit();
});
        closeWindow.setAccelerator(new KeyCodeCombination(KeyCode.Q,
key));
        fileMenu.getItems().addAll(newTab, closeTab, closeWindow);

        // Edit
        Menu editMenu = new Menu("Edit");
        MenuItem undo = new MenuItem("Undo");
        undo.setOnAction((ActionEvent event) -> {
            // TODO
            System.out.println("Undo");
        });
        undo.setAccelerator(new KeyCodeCombination(KeyCode.Z, key));

        MenuItem redo = new MenuItem("Redo");
        redo.setOnAction((ActionEvent event) -> {
            // TODO
            System.out.println("Redo");
        });
        redo.setAccelerator(new KeyCodeCombination(KeyCode.Z, key,
KeyCombination.SHIFT_DOWN));

        editMenu.getItems().addAll(undo, redo);
        menuBar.getMenus().addAll(fileMenu, editMenu);
    }
}
############################
# EndVisitController.java #
############################

package lbms.controllers.guicontrollers.VisitControllers;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * EndVisitController class for the Library Book Management System.
 * @author Team b
 */
```

```
public class EndVisitController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private TextField visitorIdField;
    @FXML private Text visitorIdFail;
    @FXML private Text failedLabel;

    /**
     * Initializes the state for this class.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                end();
                e.consume();
            }
        });
    }

    /**
     * Setter for the session manager.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Ends the visit to the library.
     */
    @FXML
    public void end() {
        this.visitorIdFail.setText("");
        this.failedLabel.setText("");

        String visitorId = this.visitorIdField.getText();

        if (visitorId.isEmpty()) {
            this.visitorIdFail.setText("*");
            this.failedLabel.setText("Please enter a visitor ID.");
        } else {
            String request = String.format("%s,depart,%s;",
this.manager.getClientId(), visitorId);
            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("invalid-id")) {
                this.failedLabel.setText("Visitor is currently not in the
library.");
            } else {
                this.manager.display("visit_ended", "Visit Ended");

((VisitEndedController)this.manager.getController()).setVisit(responseObj
ect);
```

```
            }
        }
    }

    /**
     * Cancels the request and loads the employee main page.
     */
    @FXML
    public void cancel() {
        this.manager.display("main_employee", this.manager.getUser());
    }
}
############################
# VisitEndedController.java #
############################

package lbms.controllers.guicontrollers.VisitControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * VisitEndedController class for the Library Book Management System.
 * @author Team B
 */
public class VisitEndedController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private Text visitorID;
    @FXML private Text visitEndTime;
    @FXML private Text visitDuration;

    /**
     * Initializes the state for this class.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                home();
                e.consume();
            }
        });
    }

    /**
     * Setter for the session manager.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
```

```
        this.manager = manager;
    }

    /**
     * Goes to the home page for an employee.
     */
    @FXML
    public void home() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Sets the visit.
     * @param response: input from the parse response utility
     */
    @FXML
    public void setVisit(HashMap<String, String> response) {
        this.visitorID.setText(response.get("visitorID"));
        this.visitEndTime.setText(response.get("visitEndTime"));
        this.visitDuration.setText(response.get("visitDuration"));
    }
}

############################
# VisitBegunController.java #
############################

package lbms.controllers.guicontrollers.VisitControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * VisitBegunController class for the Library Book Management System.
 * @author Team B
 */
public class VisitBegunController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private Text visitorID;
    @FXML private Text visitDate;
    @FXML private Text visitStartTime;

    /**
     * Initializes the state for this class.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                home();
```

```
                e.consume();
            }
        });
    }

    /**
     * Setter for the session manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Goes back to the home page.
     */
    @FXML
    public void home() {
        this.manager.display("main_employee", this.manager.getUser());
    }

    /**
     * Sets the visit in the library book management system.
     * @param response: input from the parse response utility
     */
    @FXML
    public void setVisit(HashMap<String, String> response) {
        this.visitorID.setText(response.get("visitorID"));
        this.visitDate.setText(response.get("visitDate"));
        this.visitStartTime.setText(response.get("visitStartTime"));
    }
}

############################
# BeginVisitController.java #
############################

package lbms.controllers.guicontrollers.VisitControllers;

import javafx.fxml.FXML;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * BeginVisitController class for the Library Book Management System.
 * @author Team B
 */
public class BeginVisitController implements StateController {

    private SessionManager manager;
```

```
    @FXML private AnchorPane root;
    @FXML private TextField visitorIdField;
    @FXML private Text visitorIdFail;
    @FXML private Text failedLabel;

    /**
     * Initializes the state for this instance of the class.
     */
    @FXML
    protected void initialize() {
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                begin();
                e.consume();
            }
        });
    }

    /**
     * Sets the session manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Begins the visit.
     */
    @FXML
    public void begin() {
        this.visitorIdFail.setText("");
        this.failedLabel.setText("");

        String visitorId = this.visitorIdField.getText();

        if (visitorId.isEmpty()) {
             this.visitorIdFail.setText("*");
             this.failedLabel.setText("Please enter a visitor ID.");
        } else {
            String request = String.format("%s,arrive,%s;",
this.manager.getClientId(), visitorId);
            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            switch (responseObject.get("message")) {
                case "invalid-id":
                    this.failedLabel.setText("Visitor does not exist.");
                    break;
                case "library-closed":
                    this.failedLabel.setText("Sorry the library is
closed, please try again later.");
                    break;
                case "duplicate":
                    this.failedLabel.setText("Visitor is already in the
library.");
```

```
                default:
                    this.manager.display("visit_begun", "Visit Begun");

((VisitBegunController)this.manager.getController()).setVisit(responseObj
ect);
                    break;
            }
        }
    }

    /**
     * Cancels the request and loads the main employee page.
     */
    @FXML
    public void cancel() {
        this.manager.display("main_employee", this.manager.getUser());
    }
}

########################
# LoginController.java #
########################

package lbms.controllers.guicontrollers;

import javafx.fxml.FXML;
import javafx.scene.control.Button;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * StateController class used for the state of the GUI.
 * @author Team B
 */
public class LoginController implements StateController {

    private SessionManager manager;

    @FXML private AnchorPane root;
    @FXML private TextField usernameField;
    @FXML private PasswordField passwordField;
    @FXML private Text loginFailedLabel;
    @FXML private Button loginButton;
    @FXML private Text usernameFail;
    @FXML private Text passwordFail;

    /**
     * Initializes the login page.
     */
    @FXML
    protected void initialize() {
```

```
        this.root.addEventHandler(KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                this.loginButton.fire();
                e.consume();
            }
        });
    }

    /**
     * Initializes the manager.
     * @param manager: the session manager to be set
     */
    public void initManager(final SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Executes the controller.
     */
    @FXML
    private void execute() {
        this.usernameFail.setText("");
        this.passwordFail.setText("");
        boolean completed = true;

        if (this.usernameField.getText().isEmpty()) {
            this.usernameFail.setText("*");
            completed = false;
        }
        if (this.passwordField.getText().isEmpty()) {
            this.passwordFail.setText("*");
            completed = false;
        }

        if (completed) {
            try {
                String request = String.format("%s,login,%s,%s;",
                        this.manager.getClientId(),
this.usernameField.getText(), this.passwordField.getText());
                String response = new
ProxyCommandController().processRequest(request);

                HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
                if (responseObject.get("message").equals("success")) {

this.manager.setVisitor(ProxyCommandController.getVisitorID(this.manager.
getClientId()));
                    this.manager.setUser(this.usernameField.getText());

                    if
(ProxyCommandController.isEmployee(this.manager.getClientId())) {
                        this.manager.display("main_employee",
this.manager.getUser());
                    } else {
                        this.manager.display("main_visitor",
this.manager.getUser());
                    }
                } else {
                    throw new Exception();
```

```
                }
            } catch (Exception e) {
                this.loginFailedLabel.setText("Invalid Username or
Password. Please Try Again.");
            }
        } else {
            this.loginFailedLabel.setText("* Please enter missing
fields.");
        }
    }
}

################################
# PaymentSuccessController.java #
################################

package lbms.controllers.guicontrollers.ReturnControllers;

import javafx.fxml.FXML;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.AnchorPane;
import javafx.scene.text.Text;
import javafx.stage.Stage;

/**
 * Created by Chris on 4/18/17.
 */
public class PaymentSuccessController {
    @FXML private AnchorPane root;
    @FXML private Text title, visitor, paid, balance;

    @FXML protected void initialize() {

this.root.addEventHandler(javafx.scene.input.KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                close();
                e.consume();
            }
        });
    }

    void load(String visitor, String payment, String balance) {
        this.visitor.setText(visitor);
        this.paid.setText("$" +payment);
        this.balance.setText("$" +balance);
    }

    @FXML
    public void close() {
        Stage stage = (Stage)title.getScene().getWindow();
        stage.close();
    }
}

############################
# ReturnBookController.java #
############################

package lbms.controllers.guicontrollers.ReturnControllers;
```

```
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.CheckBox;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

import java.util.ArrayList;
import java.util.HashMap;

/**
 * ReturnBookController class for the library book management system.
 * @author Team B
 */
public class ReturnBookController implements StateController {

    private SessionManager manager;
    private HashMap<CheckBox, String> options = new HashMap<>();
    private ArrayList<HashMap<String, String>> books;
    private String visitor;

    @FXML private VBox results;
    @FXML private TextField visitorIdField;
    @FXML private Text failedLabel, visitorIdFail;

    /**
     * Initializes the state for this instance of the class.
     */
    @FXML
    protected void initialize() {

this.results.getParent().addEventHandler(javafx.scene.input.KeyEvent.KEY_
PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                find();
                e.consume();
            }
        });
    }

    /**
     * Initializes the manager for this instance of the class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Finds books that are not returned.
     */
```

```
    @FXML
    public void find() {
        this.visitorIdFail.setText("");
        this.failedLabel.setText("");
        this.results.getChildren().clear();

        this.visitor = this.visitorIdField.getText();

        if (this.visitor.isEmpty()) {
            this.visitorIdFail.setText("*");
            this.failedLabel.setText("Please enter a visitor ID.");
        } else {
            String request = this.manager.getClientId() + ",borrowed," +
this.visitor + ";";
            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("success")) {
                if (Integer.parseInt(responseObject.get("numberOfBooks"))
== 0) {
                    this.failedLabel.setText("This visitor has not
borrowed any books.");
                } else {
                    this.books =
ParseResponseUtility.parseBooks(responseObject.get("books"));

                    for (HashMap<String, String> book: this.books) {
                        try {
                            FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/borrowed.fxml"
));

this.results.getChildren().add(loader.load());
                            BorrowedResultController controller =
loader.getController();
                            controller.load(this.manager, book);
                            this.options.put(controller.getCheckBox(),
book.get("id"));
                        } catch (Exception e) {
                            System.out.println("Error loading book.");
                        }
                    }
                }
            } else {
                this.failedLabel.setText("Invalid visitor ID. Please try
again.");
            }

        }
    }

    /**
     * Returns books through the GUI.
     */
    @FXML
    void returnBooks() {
```

```
        String request = this.manager.getClientId() + ",return," +
this.visitor + ",{";
        if (this.options.isEmpty()) {
            this.failedLabel.setText("This visitor has no borrowed
books.");
        } else {
            for (CheckBox box: this.options.keySet()) {
                if (box.isSelected()) {
                    request += this.options.get(box) + ",";
                }
            }
            request = request.substring(0, request.lastIndexOf(",")) +
"};";

            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("success")) {
                this.manager.display("return_success", "Book Returned");
            } else if (responseObject.get("message").equals("overdue")){
                Parent root;
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/payment.fxml")
);
                    root = loader.load();

                    Stage stage = new Stage();
                    stage.setTitle(this.visitor + " - Pay Fine");
                    stage.setScene(new Scene(root, 750, 500));


((PayFineController)loader.getController()).load(stage, this.manager,
this.visitor, responseObject,
                            this.books);
                    stage.show();
                }
                catch (Exception e) {
                    System.out.println("Error loading FXML file.");
                    System.exit(1);
                }
            } else {
                this.failedLabel.setText("Error");
            }

        }
    }

    /**
     * Cancels anything on this page and goes back a page.
     */
    @FXML
    public void cancel() {
        this.manager.display("main_employee", this.manager.getUser());
    }
}
```

```
################################
# BorrowedResultController.java #
################################

package lbms.controllers.guicontrollers.ReturnControllers;

import javafx.fxml.FXML;
import javafx.scene.control.CheckBox;
import javafx.scene.text.Text;
import lbms.views.GUI.SessionManager;

import java.util.HashMap;

/**
 * BorrowedResultController class for the Library Book Management System.
 * @author Team B
 */
public class BorrowedResultController {

    private SessionManager manager;
    private HashMap<String, String> book;

    @FXML private CheckBox checkBox;
    @FXML private Text isbn, title, date;

    /**
     * Loads the data for this class.
     * @param manager: the session manager
     * @param book: the hash map of books that are borrowed
     */
    public void load(SessionManager manager, HashMap<String, String>
book) {
        this.manager = manager;
        this.book = book;
        populate();
    }

    /**
     * Populates the list of borrowed books.
     */
    private void populate() {
        this.isbn.setText(this.book.get("isbn"));
        this.title.setText(this.book.get("title"));
        this.date.setText(this.book.get("dateBorrowed"));
    }

    /**
     * Getter for the check box.
     * @return the check box
     */
    CheckBox getCheckBox() {
        return this.checkBox;
    }
}

################################
# BookReturnedController.java #
################################

package lbms.controllers.guicontrollers.ReturnControllers;
```

```
import javafx.fxml.FXML;
import lbms.controllers.guicontrollers.StateController;
import lbms.views.GUI.SessionManager;

/**
 * BookReturnedController class for the Library Book Management System.
 * @author Team B
 */
public class BookReturnedController implements StateController {

    private SessionManager manager;

    /**
     * Initializes the manager for this class.
     * @param manager: the session manager to be set
     */
    @Override
    public void initManager(SessionManager manager) {
        this.manager = manager;
    }

    /**
     * Loads the home page for the employee that is logged in.
     */
    @FXML
    public void home() {
        this.manager.display("main_employee", this.manager.getUser());
    }
}
```

```
##########################
# PayFineController.java #
##########################
```

```
package lbms.controllers.guicontrollers.ReturnControllers;

import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.TextField;
import javafx.scene.input.KeyCode;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.VBox;
import javafx.scene.text.Text;
import javafx.stage.Stage;
import lbms.controllers.commandproxy.ParseResponseUtility;
import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.views.GUI.SessionManager;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

/**
 * PayFineController class for the Library Book Management System.
 * @author Team B
 */
```

```
public class PayFineController {

    private SessionManager manager;
    private Stage stage;
    private String visitor;
    private HashMap<String, String> response;
    private ArrayList<HashMap<String, String>> books;

    @FXML private AnchorPane root;
    @FXML private VBox results;
    @FXML private TextField input;
    @FXML private Text title, fine, failedLabel;

    /**
     * Initializes the data for this class.
     */
    @FXML
    protected void initialize() {

this.root.addEventHandler(javafx.scene.input.KeyEvent.KEY_PRESSED, e -> {
            if (e.getCode() == KeyCode.ENTER) {
                pay();
                e.consume();
            }
        });
    }


    /**
     * Pays the fine.
     */
    @FXML
    public void pay() {
        String payment = this.input.getText();

        if (payment.isEmpty()) {
            this.failedLabel.setText("Please enter and amount to pay.");
        } else {
            String request = this.manager.getClientId() + ",pay," +
payment + "," + this.visitor + ";";
            String response = new
ProxyCommandController().processRequest(request);

            HashMap<String, String> responseObject =
ParseResponseUtility.parseResponse(response);
            if (responseObject.get("message").equals("success")) {
                try {
                    FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/payment_succes
s.fxml"));
                    Parent root = loader.load();

((PaymentSuccessController)loader.getController()).load(this.visitor,
payment,
                            responseObject.get("balance"));
                    this.stage.setScene(new Scene(root, 750, 500));
                }
                catch (Exception e) {
                    e.printStackTrace();
```

```
                }
            } else {
                this.failedLabel.setText("Invalid amount. Please try
again.");
            }
        }
    }

    /**
     * Loads the data for this class and displays it.
     * @param stage: the stage
     * @param manager: the session manager
     * @param visitor: the visitor of the tab
     * @param response: the parse response data
     * @param books: the books with fines
     */
    void load(Stage stage, SessionManager manager, String visitor,
HashMap<String, String> response,
              ArrayList<HashMap<String, String>> books) {
        this.stage = stage;
        this.manager = manager;
        this.response = response;
        this.visitor = visitor;
        this.books = books;
        display();
    }

    /**
     * Displays the data for this class.
     */
    private void display() {
        this.title.setText("Visitor " + this.visitor + " has overdue
books.\nPlease pay fines to continue.");
        this.fine.setText(this.response.get("fine"));

        List<String> ids =
Arrays.asList(this.response.get("ids").split("\\s*,\\s*"));
        for (String id: ids) {
            for (HashMap<String, String> book: this.books) {
                if (book.get("id").equals(id)) {
                    try {
                        FXMLLoader loader = new FXMLLoader();

loader.setLocation(SessionManager.class.getResource("/fxml/overdue.fxml")
);
                        this.results.getChildren().add(loader.load());
                        BorrowedResultController controller =
loader.getController();
                        controller.load(this.manager, book);
                    } catch (Exception e) {
                        System.out.println("Error loading book.");
                    }
                    break;
                }
            }
        }
    }
}

##############################
```

```
# ProxyCommandController.java #
##############################

package lbms.controllers.commandproxy;

import lbms.LBMS;
import lbms.command.Invalid;
import lbms.models.*;

import java.time.LocalTime;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

/**
 * Implementation of a protection proxy to control access
 * to CommandController based on access rights.
 * @author Team B
 */
public class ProxyCommandController implements ICommandController {

    private static LibraryState libraryStatus = null;

    /**
     * Checks to ensure the command is being requested by the proper user
     * before sending the command to the CommandController.
     * @param requestString the input string to be processed
     * @return the response string
     */
    public String processRequest(String requestString) {
        if (requestString.charAt(requestString.length() - 1) != ';' &&
!requestString.equals("quit") &&
                !requestString.equals("exit")) {
            return "partial-request;";
        }
        String request[] = requestString.replace(";", "").split(",", 3);
        if (request[0].equals("connect")) {
            return new CommandController().processRequest(requestString);
        }

        long clientID;
        try {
            clientID = Long.parseLong(request[0]);
        } catch (NumberFormatException e) {
            if (!requestString.equals("quit") &&
!requestString.equals("exit")) {
                return "invalid-client-id;";
            }
            return "";
        }
        String command = request[1];

        // allows users who are not logged in to only perform specific
actions
        if (loginRequired(command) && !isLoggedIn(clientID)) {
            return "not-authorized;";
        }

        if (!isCommand(command)) {
            return new Invalid().execute();
```

```
        }

        if (unrestricted(command) || (isLoggedIn(clientID) &&
isEmployee(clientID))) {
            return new CommandController().processRequest(requestString);
        } else {
            return request[0] + "," + request[1] + "," + "not-
authorized;";
        }
    }

    /**
     * Most commands can only be executed by employees while some
commands
     * can be accessed by any visitor and are therefore "unrestricted"
     * @param command the word from the request string which
differentiates the command requested
     * @return true if any visitor can perform the command, false
otherwise
     */
    private boolean unrestricted(String command) {
        ArrayList<String> visitorCommands = new
ArrayList<>(Arrays.asList(
                "arrive", "info", "borrow", "depart", "login", "logout",
"undo", "redo", "disconnect", "search"
        ));
        return visitorCommands.contains(command);
    }

    /**
     * Used to determine if a string is a valid command or not.
     * @param command: the string being checked
     * @return true if it is a command, false if not
     */
    private boolean isCommand(String command) {
        ArrayList<String> commands = new ArrayList<>(Arrays.asList(
                "register", "arrive", "depart", "info", "borrow",
"borrowed", "return", "pay", "search", "buy",
                "advance", "datetime", "report", "connect", "disconnect",
"create", "login", "logout", "undo", "redo",
                "service"
        ));
        return commands.contains(command);
    }

    /**
     * Used to determine if a command can be executed without being
logged in.
     * @param command the command to check
     * @return true if login is required to execute the command, false
otherwise
     */
    private boolean loginRequired(String command) {
        ArrayList<String> allowedCommands = new
ArrayList<>(Arrays.asList(
                "connect", "disconnect", "login", "logout"
        ));
        return !allowedCommands.contains(command);
    }
```

```
    /**
     * A particular client id always maps to a visitor but that visitor
     * may also be an employee. This function serves as an employee
check.
     * @param clientID the id of the client
     * @return true if the client id represents and employee, false
otherwise
     */
    public static boolean isEmployee(long clientID) {
        try {
            Visitor v = LBMS.getSessions().get(clientID).getV();
            for (Employee employee: LBMS.getEmployees().values()) {
                if (employee.getVisitor().getVisitorID() ==
v.getVisitorID()) {
                    return true;
                }
            }
            return false;
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Determines if someone is logged into a session/client.
     * @param clientID: the ID of the client
     * @return true if someone is logged in, false if not
     */
    private static boolean isLoggedIn(long clientID) {
        try {
            Session s = LBMS.getSessions().get(clientID);
            Visitor v = s.getV();
            return v != null;
        } catch (Exception e) {
            return false;
        }
    }

    /**
     * Determines if the visitor is in the library.
     * @param clientID: the id of the client where they are logged
     * @return true if they are in the library, false if not
     */
    public static boolean inLibrary(long clientID) {
        try {
            HashMap<Long, Visit> visits = LBMS.getCurrentVisits();
            Session s = LBMS.getSessions().get(clientID);
            for (Visit v: visits.values()) {
                if (v.getVisitor().getVisitorID() ==
s.getV().getVisitorID()) {
                    return true;
                }
            }
            return false;
        } catch (Exception e) {
            return false;
        }
    }

    /**
```

```
     * Updates the status of the library.
     */
    private static void updateStatus() {
        LocalTime time = SystemDateTime.getInstance(null).getTime();
        if (time.isAfter(LBMS.OPEN_TIME) &&
time.isBefore(LBMS.CLOSE_TIME)) {
            libraryStatus  = new OpenState();
        } else {
            libraryStatus = new ClosedState();
        }
    }

    /**
     * Checks if the library is currently open based on the system time
     * @return true if the library is open, false otherwise
     */
    public static boolean isOpen() {
        updateStatus();
        return libraryStatus.isOpen();
    }

    /**
     * Employees are able to perform actions for visitors and therefore
may
     * input a visitorID which does not match their clientID.
     * Visitors are unable to due this for other visitors.
     * @param visitorID id of visitor to perform action on
     * @param clientID id of client requesting action
     * @return true if clientID can perform action for visitorID, false
otherwise
     */
    public static boolean assistanceAuthorized(long visitorID, long
clientID) {
        return visitorID ==
LBMS.getSessions().get(clientID).getV().getVisitorID() ||
                isEmployee(clientID);
    }

    /**
     * Getter for the visitor ID when the client ID is known.
     * @param client: the client ID of the session
     * @return the visitor ID of who is logged in
     */
    public static Long getVisitorID(Long client) {
        return LBMS.getSessions().get(client).getV().getVisitorID();
    }

    /**
     * Getter for the store that is currently being searched.
     * @return the search service
     */
    public static String getStore(long clientID) {
        if(LBMS.getSessions().get(clientID).getSearch() ==
LBMS.SearchService.GOOGLE) {
            return "google";
        } else {
            return "local";
        }
    }
}
```

```
#########################
# CommandController.java #
#########################

package lbms.controllers.commandproxy;

import lbms.LBMS;
import lbms.command.*;
import lbms.models.SystemDateTime;
import lbms.models.Visitor;

import java.time.LocalDateTime;

/**
 * ICommandController class interacts with the command package to execute
commands.
 * @author Team B
 */
public class CommandController implements ICommandController {

    private static Command command = null;

    /**
     * Takes in a request string and outputs a response string.
     * @param requestString: the input string to be processed
     * @return the response output string
     */
    public String processRequest(String requestString) {
        String response = "";
        if (requestString.endsWith(";")) {
            String request[] = requestString.replace(";", "").split(",",
3);
            try {
                command = createCommand(request);
                if (request[0].equals("connect")) {
                    response = request[0] + "," + command.execute() +
";";
                } else {
                    response = request[0] + "," + request[1] +
command.execute();
                }
            } catch (MissingParametersException e) {
                response = request[0] + "," + request[1] + "," +
e.getMessage();
            } catch (Exception e) {
                response = request[0] + "," + request[1] + ",missing-
parameters,{all};";
            }
        } else if (!requestString.equals("exit")) {
            response = "partial-request;";
        }

        return response;
    }

    /**
     * Getter for the command.
     * @return the command
     */
```

```
    public static Command getCommand() {
        return command;
    }

    /**
     * Getter for the system clock.
     * @return a local date time of the system clock
     */
    public static LocalDateTime getSystemDateTime() {
        return SystemDateTime.getInstance(null).getDateTime();
    }

    /**
     * Creates a command based on the input request.
     * @param request: the input request to be processed
     * @return a Command object for the request
     */
    private static Command createCommand(String[] request) throws
Exception {
        if (request[0].equals("connect")) {
            return new ClientConnect();
        } else {
            long clientID = Long.parseLong(request[0]);
            if (LBMS.getSessions().get(clientID) == null) {
                throw new MissingParametersException("invalid-client-
id;");
            }

            switch (request[1]) {
                case "disconnect":
                    return new Disconnect(Long.parseLong(request[0]));
                case "create":
                    return new CreateAccount(request[2]);
                case "login":
                    return new LogIn(clientID + "," + request[2]);
                case "logout":
                    return new LogOut(clientID);
                case "undo":
                    return new Undo(clientID);
                case "redo":
                    return new Redo(clientID);
                case "service":
                    return new SetBookService(clientID, request[2]);
                case "arrive":
                    if (ProxyCommandController.isOpen()) {
                        Visitor v =
LBMS.getSessions().get(clientID).getV();
                        if (v == null) {
                            throw new MissingParametersException("not-
logged-in;");
                        }
                        BeginVisit b;

                        if (request.length == 2) {
                            b = new BeginVisit(Long.toString(clientID));
                        } else {
                            b = new BeginVisit(clientID + "," +
request[2]);
                        }
```

```
                        LBMS.getSessions().get(clientID).addUndoable(b);
                        return b;
                    }
                    return new CloseLibrary();
                case "borrow":
                    if (ProxyCommandController.isOpen()) {
                        Borrow b = new Borrow(clientID + "," +
request[2]);
                        LBMS.getSessions().get(clientID).addUndoable(b);
                        return b;
                    }
                    return new CloseLibrary();
                case "register":
                    return new RegisterVisitor(request[2]);
                case "depart":
                    Visitor v = LBMS.getSessions().get(clientID).getV();
                    if (v == null) {
                        throw new MissingParametersException("not-logged-
in;");
                    }
                    EndVisit ev;

                    if (request.length == 2) {
                        ev = new EndVisit(Long.toString(clientID));
                    } else {
                        ev = new EndVisit(clientID + "," + request[2]);
                    }

                    LBMS.getSessions().get(clientID).addUndoable(ev);
                    return ev;
                case "info":
                    LBMS.getSessions().get(clientID).clearStacks();
                    return new LibrarySearch(clientID, request[2]);
                case "borrowed":
                    LBMS.getSessions().get(clientID).clearStacks();
                    if (request.length == 2) {
                        return new FindBorrowed(request[0]);
                    } else if (request.length == 3) {
                        return new FindBorrowed(request[0] + "," +
request[2]);
                    }
                case "return":
                    if (request.length == 3) {
                        Return r = new Return(request[0] + "," +
request[2]);
                        LBMS.getSessions().get(clientID).addUndoable(r);
                        return r;
                    } else if (request.length == 4) {
                        Return r = new Return(request[0] + "," +
request[2] + "," + request[3]);
                        LBMS.getSessions().get(clientID).addUndoable(r);
                        return r;
                    }
                case "pay":
                    PayFine pf;
                    if (request.length == 3) {
                        pf = new PayFine(request[0] + "," + request[2]);
                        LBMS.getSessions().get(clientID).addUndoable(pf);
                        return pf;
                    } else if (request.length == 4) {
```

```
                        pf = new PayFine(request[0] + "," + request[2] +
"," + request[3]);
                        LBMS.getSessions().get(clientID).addUndoable(pf);
                        return pf;
                    }
                case "search":
                    LBMS.getSessions().get(clientID).clearStacks();
                    return new StoreSearch(clientID, request[2]);
                case "buy":
                    BookPurchase bp = new BookPurchase(clientID,
request[2]);
                    LBMS.getSessions().get(clientID).addUndoable(bp);
                    return bp;
                case "advance":
                    return new AdvanceTime(request[2]);
                case "datetime":
                    return new GetDateTime();
                case "report":
                    if (request.length == 2) {
                        return new StatisticsReport("");
                    }
                    return new StatisticsReport(request[2]);
                case "reset":      // FOR TESTING ONLY
                    return new ResetTime();
                default:
                    return new Invalid();
            }
        }
    }
}
```

```
############################
# ParseResponseUtility.java #
############################
```

```
package lbms.controllers.commandproxy;

import lbms.models.Book;
import lbms.search.BookSearch;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;

/**
 * Utility class for parsing responses.
 * @author Team B
 */
public final class ParseResponseUtility {

    /**
     * Private constructor to prevent any instantiation of this class.
     */
    private ParseResponseUtility() {}

    /**
     * Takes in a response string and parses it into a HashMap.
     * The HashMap returned maps strings (eg. authors) to strings from
the response.
     * Formatting Notes:
```

```
     *     --All HashMap keys are camelCased
     *     --The name of the command has the key "command"
     *     --Error messages and the success message have the key
"message"
     *     --more information about the "message" has the key
"invalidValue" (if applicable)
     *     --if the value returned is a large string of books, use
parseBooks
     *     --the report from the report command is not parsed
     * @param response response string from a command
     * @return HashMap of the parsed response
     */
    public static HashMap<String, String> parseResponse(String response)
{
        HashMap<String, String> parsed = new HashMap<>();

        String[] fields = response.replace(";", "").replace("\n",
"\nBOOK:").split("[,\\n]");

        if (fields[0].equals("connect")) {
            parsed.put("command", fields[0]);
            parsed.put("clientID", fields[1]);
        } else {
            parsed.put("clientID", fields[0]);
            parsed.put("command", fields[1]);

            if (fields.length > 2 && isErrorMessage(fields[2])) {
                parsed.put("message", fields[2]);
                String invalidValue = ""; // invalid value may be a set
of values

                for (int i = 3; i < fields.length; i++) {
                    if (invalidValue.length() != 0) {
                        invalidValue += ",";
                    }
                    invalidValue += fields[i];
                }

                parsed.put("invalidValue", invalidValue);
            } else {
                switch (fields[1]) {
                    case "advance":
                        parsed.put("message", fields[2]);
                        break;
                    case "arrive":
                        parsed.put("message", "success");
                        parsed.put("visitorID", fields[2]);
                        parsed.put("visitDate", fields[3]);
                        parsed.put("visitStartTime", fields[4]);
                        break;
                    case "borrow":
                        parsed.put("message", "success");
                        parsed.put("dueDate", fields[2]);
                        break;
                    case "borrowed":
                        parsed.put("message", "success");
                        parsed.put("numberOfBooks", fields[2]);
                        parsed.put("books", removeBooks(3,fields));
                        break;
                    case "buy":
```

```
                        parsed.put("message", fields[2]);
                        parsed.put("numberOfBooks", fields[3]);
                        parsed.put("books", removeBooks(4, fields));
                        break;
                    case "create":
                        parsed.put("message", fields[2]);
                        break;
                    case "datetime":
                        parsed.put("message", "success");
                        parsed.put("date", fields[2]);
                        parsed.put("time", fields[3]);
                        break;
                    case "depart":
                        parsed.put("message", "success");
                        parsed.put("visitorID", fields[2]);
                        parsed.put("visitEndTime", fields[3]);
                        parsed.put("visitDuration", fields[4]);
                        break;
                    case "disconnect":
                        parsed.put("message", "success");
                        break;
                    case "info":
                        parsed.put("message", "success");
                        parsed.put("numberOfBooks", fields[2]);
                        parsed.put("books", removeBooks(3,fields));
                        break;
                    case "login":
                    case "logout":
                        parsed.put("message", fields[2]);
                        break;
                    case "pay":
                        parsed.put("message", fields[2]);
                        parsed.put("balance",fields[3]);
                        break;
                    case "redo":
                        parsed.put("message", fields[2]);
                        break;
                    case "register":
                        parsed.put("message", "success");
                        parsed.put("visitorID", fields[2]);
                        parsed.put("registrationDate", fields[3]);
                        break;
                    case "report":
                        String[] reportFields = response.replace(";",
"").split(",");
                        parsed.put("message", "success");
                        parsed.put("date",reportFields[2]);
                        parsed.put("report",reportFields[3]);
                        break;
                    case "reset":
                        parsed.put("message",fields[2]);
                        break;
                    case "return":
                        parsed.put("message",fields[2]);
                        if (fields.length > 3) {
                            parsed.put("fine",fields[3]);
                            String ids = "";

                            for (int i = 4; i < fields.length; i++) {
                                if (ids.length() != 0) {
```

```
                                ids += ",";
                            }
                            ids += fields[i];
                        }

                        parsed.put("ids", ids); // comma separated
list of ids
                    }
                    break;
                case "search":
                    parsed.put("message", "success");
                    parsed.put("numberOfBooks", fields[2]);
                    parsed.put("books", removeBooks(3,fields));
                    break;
                case "service":
                    parsed.put("message", fields[2]);
                    break;
                case "undo":
                    parsed.put("message", fields[2]);
                    break;
                default:
                    // remember to check for not-authorized
                    parsed.put("message", "failure");
                    System.out.println("Bad response to parse");
                    parsed.put("badResponse", response);
                    break;
            }
        }
    }

        return parsed;
    }

    /**
     * Takes an arbitrarily long string of books and creates an ArrayList
with an
     * entry for each book.
     * Each index is a HashMap mapping string of types of Book
information (eg. title)
     * to a string of the associated value.
     * IMPORTANT: this function was only meant to parse strings that were
     *            part of a command response
     * @return ArrayList(HashMap[bookInfoType, bookInfo])
     */
    public static ArrayList<HashMap<String,String>> parseBooks(String
booksString) {
        ArrayList<HashMap<String,String>> books = new ArrayList<>();

        String[] booksArray = booksString.replaceAll("^BOOK:",
"").split("BOOK:");

        for (String book: booksArray) {
            String[] bookPieces = book.replace("BOOK:", "").split(",");
            String publishDate = "";
            for (String piece: bookPieces) {
                if (piece.matches("\\d{2}/\\d{2}/\\d{4}")) {
                    publishDate = piece;
                }
            }
            HashMap<String,String> bookInfo = new HashMap<>();
```

```
            if (bookPieces.length == 4) { //FindBorrowed
                bookInfo.put("id", bookPieces[0]);
                bookInfo.put("isbn", bookPieces[1]);
                bookInfo.put("title", bookPieces[2]);
                bookInfo.put("dateBorrowed", bookPieces[3]);
                books.add(bookInfo);
            } else if (bookPieces[0].length() >= 10) { // buy
                bookInfo.put("isbn", bookPieces[0]);
                bookInfo.put("title", bookPieces[1].replaceAll("\"",
"""));
                String authorString = "";
                for(int index = 2; index < bookPieces.length-2; index++)
{
                    authorString += bookPieces[index].replaceAll("[{}]",
"");
                    if(index != bookPieces.length-3) {
                        authorString += ",";
                    }
                }
                bookInfo.put("authors", authorString);
                bookInfo.put("publishDate", publishDate);
                bookInfo.put("quantity", bookPieces[bookPieces.length-
1]);
                books.add(bookInfo);
            } else if (bookPieces[1].length() > 3) { // search
                bookInfo.put("id", bookPieces[0]);
                if(BookSearch.BY_ISBN.toBuy().findFirst(bookPieces[1]) !=
null) {
                    Book b =
BookSearch.BY_ISBN.toBuy().findFirst(bookPieces[1]);
                    bookInfo.put("isbn", b.getIsbn().toString());
                    bookInfo.put("title", b.getTitle());
                    bookInfo.put("authors", b.getAuthorsString());
                    bookInfo.put("publishDate", publishDate);
                    bookInfo.put("publisher", b.getPublisher());
                    bookInfo.put("pageCount", b.getPageCount() + "");
                    books.add(bookInfo);
                }
                else { // from GoogleAPI
                    bookInfo.put("isbn", bookPieces[1]);
                    bookInfo.put("title", bookPieces[2].replaceAll("\"",
"""));
                    String authorString = "";
                    for(int index = 3; index < bookPieces.length-2;
index++) {
                        authorString +=
bookPieces[index].replaceAll("[{}]", "");
                        if(index != bookPieces.length-3) {
                            authorString += ",";
                        }
                    }
                    bookInfo.put("authors", authorString);
                    bookInfo.put("publishDate",
bookPieces[bookPieces.length-2]);
                    bookInfo.put("publisher", "Unknown");
                    bookInfo.put("pageCount",
bookPieces[bookPieces.length-1]);
                    books.add(bookInfo);
                }
            } else { // info
```

```
                if(BookSearch.BY_ISBN.toBuy().findFirst(bookPieces[2]) !=
null) {
                    bookInfo.put("quantity", bookPieces[0]);
                    bookInfo.put("id", bookPieces[1]);
                    Book b =
BookSearch.BY_ISBN.toBuy().findFirst(bookPieces[2]);
                    bookInfo.put("isbn", b.getIsbn().toString());
                    bookInfo.put("title", b.getTitle());
                    bookInfo.put("authors", b.getAuthorsString());
                    bookInfo.put("publishDate", publishDate);
                    bookInfo.put("publisher", b.getPublisher());
                    bookInfo.put("pageCount", b.getPageCount() + "");
                    books.add(bookInfo);
                }
                else { // from googleAPI
                    bookInfo.put("quantity", bookPieces[0]);
                    bookInfo.put("id", bookPieces[1]);
                    bookInfo.put("isbn", bookPieces[2]);
                    bookInfo.put("title", bookPieces[3].replaceAll("\"",
""));
                    String authorString = "";
                    for(int index = 4; index < bookPieces.length-3;
index++) {
                        authorString +=
bookPieces[index].replaceAll("[{}]", "");
                        if(index != bookPieces.length-3) {
                            authorString += ",";
                        }
                    }
                    bookInfo.put("authors", authorString.replaceAll(",$",
""));
                    bookInfo.put("publisher",
bookPieces[bookPieces.length-3].replaceAll("\"", ""));
                    bookInfo.put("publishDate", publishDate);
                    bookInfo.put("pageCount",
bookPieces[bookPieces.length-1]);
                    books.add(bookInfo);
                }

            }
        }
        return books;
    }

    /**
     * Some commands provide a response ending with an arbitrary number
of books.
     * This function removes all of them from startingIndex to the end
and
     * creates one large string of them.
     * Pass this string to parseBooks for further parsing.
     * @param startIndex the index where the first book lives
     * @param fields fields from the split response string
     * @return large book string
     */
    private static String removeBooks(int startIndex, String[] fields) {
        String books = "";
        for (int i = startIndex; i < fields.length; i++) {
            if (books.length() != 0) {
                books += ",";
```

```
            }
            books += fields[i];
        }
        return books.replace(",,", ",");
    }

    /**
     * Error messages are handled fairly similarly by all command
responses.
     * This function checks if the provided message is an error response.
     * @param message the message in question
     * @return true if the message is an error response, false otherwise
     */
    private static boolean isErrorMessage(String message) {
        ArrayList<String> errorMessages = new ArrayList<>(Arrays.asList(
                "not-authorized", "invalid-number-of-days", "invalid-
number-of-hours", "duplicate", "invalid-id",
                "invalid-visitor-id", "invalid-book-id", "book-limit-
exceeded", "outstanding-fine",
                "duplicate-username", "duplicate-visitor", "invalid-sort-
order", "bad-username-or-password",
                "invalid-amount", "cannot-redo", "cannot-undo", "invalid-
visitor", "missing-parameters",
                "library-closed", "incorrect-value-for-days", "no-more-
copies"
        ));
        return errorMessages.contains(message);
    }
}

############################
# ICommandController.java #
############################

package lbms.controllers.commandproxy;

/**
 * Interface for the command controller and the proxy command controller.
 * @author Team B
 */
public interface ICommandController {

    /**
     * Processes a request string, manipulates the backend of the
program, and generates a response string.
     * @param requestString: the request string being read
     * @return a response string
     */
    String processRequest(String requestString);

}

#############
# LBMS.java #
#############

package lbms;

import lbms.controllers.commandproxy.ProxyCommandController;
import lbms.models.*;
```

```
import lbms.views.ViewFactory;

import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.*;

/**
 * Main class to run the Library Book Management System.
 * 2 different "modes": API, GUI
 * API: used for directly sending requests and receiving responses.
 * GUI: graphical-user-interface that is based on the API functionality
 *
 * Rochester Institute of Technology
 * SWEN-262 Section: 3, Team B
 * @author Charles Barber   crb7054@rit.edu
 * @author Nicholas Feldman ncf1362@rit.edu
 * @author Christopher Lim  cxl2436@rit.edu
 * @author Anthony Palumbo  ajp1925@rit.edu
 * @author Edward Wong      exw4141@rit.edu
 */
public class LBMS {

    /** StartType enum for determining how the program should be run. */
    public enum StartType { GUI, API }

    /** SearchService enum used for searching for books to buy. */
    public enum SearchService { LOCAL, GOOGLE }

    /** Constants for the opening and closing time. */
    public final static LocalTime OPEN_TIME = LocalTime.of(8, 0);
    public final static LocalTime CLOSE_TIME = LocalTime.of(19, 0);

    /** Data that is serialized on a clean exit. */
    private static HashMap<ISBN, Book> books = new HashMap<>();
    private static List<Book> booksToBuy = new ArrayList<>();
    private static HashMap<Long, Visitor> visitors = new HashMap<>();
    private static HashMap<Long, Employee> employees = new HashMap<>();
    private static List<Visit> totalVisits = new ArrayList<>();
    private static List<Transaction> transactions = new ArrayList<>();

    /** Data that is used during runtime, but not serialized. */
    private static HashMap<Long, Visit> currentVisits = new HashMap<>();
    private static HashMap<Long, Session> sessions = new HashMap<>();
    private static long totalSessions = 0;

    /**
     * Program entry point. Handle command line arguments and start.
     * @param args: the program arguments
     */
    public static void main(String[] args) {
        StartType type;
        try {
            type = StartType.valueOf(args[0].toUpperCase());
            new LBMS(type);
        } catch (ArrayIndexOutOfBoundsException e) {
            new LBMS(StartType.GUI);
        } catch (IllegalArgumentException e) {
```

```
            System.out.println("Usage: java -jar LBMS.jar <type>");
            System.out.println("Valid types are: GUI or API");
            System.exit(1);
        } finally {
            System.gc();
        }
    }

    /**
     * Handles user input for the LBMS system.
     */
    public LBMS(StartType type) {
        SystemInit();
        ViewFactory.start(type);
        SystemClose();
    }

    /**
     * Initializes the system.
     * Warnings are suppressed for reading in the serialization.
     */
    @SuppressWarnings("unchecked")
    private void SystemInit() {
        // Deserialize the data.
        try {
            FileInputStream f = new FileInputStream("data.ser");
            ObjectInputStream in = new ObjectInputStream(f);
            books = (HashMap<ISBN, Book>)in.readObject();
            booksToBuy = (ArrayList<Book>)in.readObject();
            visitors = (HashMap<Long, Visitor>)in.readObject();
            employees = (HashMap<Long, Employee>)in.readObject();
            totalVisits = (ArrayList<Visit>)in.readObject();
            transactions = (ArrayList<Transaction>)in.readObject();

SystemDateTime.getInstance((LocalDateTime)in.readObject()).start();
        } catch (ClassNotFoundException | IOException e) {
            books = new HashMap<>();
            booksToBuy = makeBooks();
            visitors = new HashMap<>();
            employees = new HashMap<>();
            totalVisits = new ArrayList<>();
            transactions = new ArrayList<>();
            SystemDateTime.getInstance(null).start();

            // Admin account creation.
            Employee employee = new Employee(new Visitor("firstname",
"lastname", "admin",
                    "password", "address", new
PhoneNumber(585,123,1234)));
            visitors.put(employee.getVisitor().getVisitorID(),
employee.getVisitor());
            employees.put(employee.getVisitor().getVisitorID(),
employee);
        }
        System.gc(); // Collects any unused data and takes out the trash!
    }

    /**
     * Serializes the data in the system for future startup.
     */
```

```
    private void SystemClose() {
        SystemDateTime.getInstance(null).stopClock();
        LibraryClose();

        // Serializes the data.
        try {
            File fl = new File("data.ser");
            FileOutputStream f = new FileOutputStream(fl);
            ObjectOutputStream out = new ObjectOutputStream(f);
            out.writeObject(books);
            out.writeObject(booksToBuy);
            out.writeObject(visitors);
            out.writeObject(employees);
            out.writeObject(totalVisits);
            out.writeObject(transactions);

out.writeObject(SystemDateTime.getInstance(null).getDateTime());
            out.close();
            f.close();
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    /**
     * Closes the library by removing all current visitors.
     */
    private static void LibraryClose() {
        // Departs all the visitors when the library closes.
        ProxyCommandController pcc = new ProxyCommandController();
        for (Visit visit: currentVisits.values()) {
            pcc.processRequest("depart," +
visit.getVisitor().getVisitorID() + ";");
        }
    }

    /**
     * Getter for the hash map of books
     * @return the books
     */
    public static HashMap<ISBN, Book> getBooks() {
        return books;
    }

    /**
     * Getter for the books to be purchased by the library.
     * @return the array list of books that can be purchased
     */
    public static List<Book> getBooksToBuy() {
        return booksToBuy;
    }

    /**
     * Getter for the visitors.
     * @return a hash map of visitors of the library
     */
    public static HashMap<Long, Visitor> getVisitors() {
        return visitors;
    }
```

```
/**
 * Getter for the employees.
 * @return a hash map of employees of the library
 */
public static HashMap<Long, Employee> getEmployees() {
    return employees;
}

/**
 * Getter for the visits made by visitors.
 * @return the array list of visits
 */
public static List<Visit> getTotalVisits() {
    return totalVisits;
}


/**
 * Getter for the currentVisits.
 * @return hash map of the current visits
 */
public static HashMap<Long, Visit> getCurrentVisits() {
    return currentVisits;
}

/**
 * Getter for the transactions.
 * @return an array list of transactions
 */
public static List<Transaction> getTransactions() {
    return transactions;
}

/**
 * Getter for the sessions.
 * @return the hash map of sessions
 */
public static HashMap<Long, Session> getSessions() {
    return sessions;
}

/**
 * Getter for the total number of sessions.
 * @return the total number of sessions
 */
public static long getTotalSessions() {
    return totalSessions;
}

/**
 * Increments the total number of sessions by one.
 */
public static void incrementSessions() {
    totalSessions++;
}

/**
 * Creates the books to be purchased from the input file.
 * @return an array list of books that the library can purchase
```

```
     */
    private ArrayList<Book> makeBooks() {
        ArrayList<Book> output = new ArrayList<>();
        try {
            InputStream inputStream =
LBMS.class.getClassLoader().getResourceAsStream("books.txt");
            Scanner s = new Scanner(inputStream);
            String[] parts;
            String line, title, publisher;
            ArrayList<String> authors;
            ISBN isbn;
            int pageCount, i;
            Calendar publishDate = null;

            while (s.hasNextLine()) {
                i = 1;
                line = s.nextLine();
                parts = line.split(",");
                isbn = new ISBN(parts[0]);
                title = "";
                authors = new ArrayList<>();
                publisher = "";

                while (parts[i].charAt(0) != '{') {
                    if (parts[i].charAt(0) == '"' &&
parts[i].charAt(parts[i].length()-1) == '"'){
                        title = parts[i].substring(1, parts[i].length()-
1);
                    } else if (parts[i].charAt(0) == '"') {
                        title = title + parts[i].substring(1) + ", ";
                    } else if (parts[i].charAt(parts[i].length()-1) ==
'"') {
                        title = title + parts[i].substring(0,
parts[i].length()-1);
                    } else {
                        title = title + parts[i].substring(1) + ",";
                    }
                    i++;
                }

                for (int in = 2; in < parts.length; in++) {
                    if (parts[in].charAt(0) == '{' &&
parts[in].charAt(parts[in].length()-1) == '}') {
                        authors.add(parts[in].substring(1,
parts[in].length()-1));
                        break;
                    } else if (parts[in].charAt(0) == '{') {
                        authors.add(parts[in].substring(1,
parts[in].length()));
                    } else if (parts[in].charAt(parts[in].length()-1) ==
'}') {
                        authors.add(parts[in].substring(0,
parts[in].length()-1));
                        break;
                    } else if (authors.size() > 0) {
                        authors.add(parts[in]);
                    }
                }

                for (int in = 3; in < parts.length; in++) {
```

```
                    if (parts[in].charAt(0) == '"' &&
parts[in].charAt(parts[in].length()-1) == '"'){
                        publisher = parts[in].substring(1,
parts[in].length()-1);
                        break;
                    } else if (parts[in].charAt(0) == '"') {
                        publisher = publisher + parts[in].substring(1) +
",";
                    } else if (parts[in].charAt(parts[in].length()-1) ==
'"' && parts[in+1].matches(".*\\d+.*")) {
                        publisher = publisher + parts[in].substring(0,
parts[in].length()-1);
                        break;
                    } else {
                        publisher = publisher + parts[in].substring(1) +
",";
                    }
                }

                try {
                    if (parts[parts.length - 2].length() == 10) {
                        SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM-dd");
                        Date date = format.parse(parts[parts.length -
2]);
                        Calendar calendar = Calendar.getInstance();
                        calendar.setTime(date);
                        publishDate = calendar;
                    } else if (parts[parts.length - 2].length() == 7) {
                        SimpleDateFormat format = new
SimpleDateFormat("yyyy-MM");
                        Date date = format.parse(parts[parts.length -
2]);
                        Calendar calendar = Calendar.getInstance();
                        calendar.setTime(date);
                        publishDate = calendar;
                    } else if (parts[parts.length - 2].length() == 4) {
                        SimpleDateFormat format = new
SimpleDateFormat("yyyy");
                        Date date = format.parse(parts[parts.length -
2]);
                        Calendar calendar = Calendar.getInstance();
                        calendar.setTime(date);
                        publishDate = calendar;
                    }
                } catch (ParseException e) {
                    e.printStackTrace();
                }

                pageCount = Integer.parseInt(parts[parts.length-1]);
                output.add(new Book(isbn, title, authors, publisher,
publishDate, pageCount, 0, 0));
            }
            inputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return output;
    }
}
```