Source Code Listings

SWEN-262: Team B
Design Project R1
3-20-2017

Command Package

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();

    /**
     * Parses the response string for standard output.
     * @param response: the response string from execute
     * @return a string for output
     */
    String parseResponse(String response);

}
```

MissingParametersException.java

```
package lbms.command;

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

    /**
     * Overloaded constructor for this exception.
     * @param message: the message for the exception.
     */
    MissingParametersException(String message) {
        super(message);
    }

}
```

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(",");
        days = Long.parseLong(arguments[0]);
        hours = arguments.length > 1 ? Long.parseLong(arguments[1]) : 0;
    }

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

    /**
     * Parses the response for advance time.
     * @param response: the response string from execute
     * @return output for advance time
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        if(fields[1].equals("success;")) {
            return "\nAdvance success, clock has been moved forward " + days + " day(s) and " + hours + " hour(s).";
        }
        else if(fields[1].equals("invalid-number-of-days")) {
            return"\nFailure, " + days + " is an invalid number of days to skip.";
        }
        else {
            return "\nFailure, " + hours + " is an invalid number of hours to skip.";
```

```
        }
    }
}
```

BeginVisit.java

```
package lbms.command;

import lbms.LBMS;
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 Command {

    private long visitorID;

    /**
     * Constructor for BeginVisit command.
     * @param request: the string that holds all the input information
     * @throws MissingParametersException: missing parameters
     */
    public BeginVisit(String request) throws MissingParametersException {
        String[] arguments = request.split(",");
        try {
            visitorID = Long.parseLong(arguments[0]);
        }
        catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
            throw new MissingParametersException("missing-parameters,visitor-ID");
        }
    }

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

        Visitor visitor = UserSearch.BY_ID.findFirst(visitorID);
        if(UserSearch.BY_ID.findFirst(visitorID).getInLibrary()) {
            return "duplicate;";
        }

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

```
    }

    /**
     * Parses the response for begin visit.
     * @param response: the response string from execute
     * @return a string for output
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        if(fields[1].equals("duplicate;")) {
            return "\nVisitor " + visitorID + " is already in the library.";
        }
        else if(fields[1].equals("invalid-id;")) {
            return "\nVisitor " + visitorID + " is not registered in the system.";
        }
        else {
            return "\nVisitor " + visitorID + " has entered the library on "
                    + fields[2] + " at " + fields[3].replace(";","") + ".";
        }
    }

    /**
     * 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;
    }
}
```

BookPurchase.java

```
package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.search.BookSearch;

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 Command {

    private int quantity;
    private List<Integer> ids;
```

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

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

/**
 * Parses the response for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    try {
        response = response.replaceAll(";$", "");
        String[] fields = response.split(",");
        if(fields[1].equals("success")) {
            String output = "Book(s) purchased, ";
            List<Book> books;
            for(int i = 2; i < fields.length; i++) {
                try {
                    books = BookSearch.BY_ISBN.search(Long.parseLong(fields[i]));
                    output += books.get(0).getTitle() + " * " + fields[1] + "\n";
                }
                catch(NumberFormatException e) {}
            }
            return output;
        }
```

```
            return null;
        }
        catch(Exception e) {
            return "failure;";
        }
    }

    /**
     * Buys *quantity* of each book listed in *ids*
     * @return a response string
     */
    private String processPurchaseOrder() {
        String booksBought = "";
        for(int id: ids) {
            Book b;
            try {
                b = LBMS.getLastBookSearch().get(id - 1);
            }
            catch(IndexOutOfBoundsException e) {
                return "failure;";
            }
            for(int i = 0; i < quantity; i++) {
                buyBook(b);
            }
            booksBought += ("\n" + b.toString() + "," + b.getNumberOfCopies()) + ",";
        }
        return 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.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;
```

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

    private long visitorID;
    private ArrayList<Integer> id;

    /**
     * Constructor for a Borrow class.
     * @param request: the request input string
     * @throws MissingParametersException: missing parameters
     */
    public Borrow(String request) throws MissingParametersException {
        String[] arguments = request.split(",");
        try {
            if(arguments.length < 2) {
                throw new NumberFormatException();
            }
            visitorID = Long.parseLong(arguments[0]);
            id = new ArrayList<>();
            for(int i = 1; i < arguments.length; i++) {
                id.add(Integer.parseInt(arguments[i]));
            }
        }
        catch(NumberFormatException e) {
            throw new MissingParametersException("missing-parameters,visitor-ID,{ids}");
        }
    }

    /**
     * Executes the borrow command.
     * @return the response or error message
     *
     * TODO make sure only x people can borrow x copies of books
     */
    @Override
    public String execute() {
        if(UserSearch.BY_ID.findFirst(visitorID) == null) {
            return "invalid-visitor-id;";
        }
        else if(UserSearch.BY_ID.findFirst(visitorID).getFines() > 0) {
            return "outstanding-fine," +
                    new DecimalFormat("#.00").format(UserSearch.BY_ID.findFirst(visitorID).getFines()) + ";";
        }
        String invalidIDs = "";
        String temp = "";
        for(Integer i: id) {
            if(!UserSearch.BY_ID.findFirst(visitorID).canCheckOut()) {
                return "book-limit-exceeded;";
            }
            temp = checkOutBook(i, visitorID);
            try {
                if(temp.contains("id-error")) {
                    String[] error = temp.split(",");
```

```
                invalidIDs += error[1];
            }
        }
        catch(NullPointerException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }
    if(invalidIDs.length() > 1) {
        String output = "invalid-book-id,";
        output += invalidIDs;
        //output = output.substring(0,output.length() - 1);
        output += ";";
        return output;
    }
    else {
        return temp + ";";
    }
}

/**
 * Parses the response for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    String[] fields = response.split(",");
    if(fields[1].equals("invalid-visitor-id;")) {
        return "\nVisitor " + visitorID + " is not registered in the system.";
    }
    else if(fields[1].equals("outstanding-fine")) {
        return "\nVisitor " + visitorID + " has to pay " +
                new DecimalFormat("#.00").format(UserSearch.BY_ID.findFirst(visitorID).getFines()) + " before they " +
                    "can borrow more books.";
    }
    else if(fields[1].equals("book-limit-exceeded;")) {
        return "\nVisitor " + visitorID + " has borrowed the maximum number of books or the borrow request would " +
                "cause the visitor to exceed 5 borrowed books.";
    }
    else if(fields[1].equals("invalid-book-id")) {
        return "\nOne of more of the book IDs specified do not match the IDs for the most recent library book search.";
    }
    else {
        return "\nThe books have been successfully borrowed and will be due on " + fields[2] + ".";
    }
}

/**
 * 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.getLastBookSearch().get(id - 1);
            t = new Transaction(b.getIsbn(), visitorID);
            v = UserSearch.BY_ID.findFirst(visitorID);
        }
        catch(Exception e) {
            return "id-error," + id;
        }

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

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;";
    }

    /**
     * Parses the response for library closed
     * @param response: the response string from execute
     * @return output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        if (fields[1].equals("library-closed;")) {
            return "\nThe library is now closed.";
        }
        return "\nThere's some kind of error";
    }
}
```

EndVisit.java

```
package lbms.command;

import lbms.LBMS;
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 Command {

    private long visitorID;

    /**
     * Constructor for an EndVisit command class.
     * @param request: the request input string
     * @throws MissingParametersException: missing parameters
     */
    public EndVisit(String request) throws MissingParametersException {
        try {
            visitorID = Long.parseLong(request);
        }
        catch(NumberFormatException e) {
            throw new MissingParametersException("missing-parameters,visitor-ID");
        }
    }

    /**
     * Executes the EndVisit command.
     * @return the response string or error message
     */
    @Override
    public String execute() {
        if(UserSearch.BY_ID.findFirst(visitorID) != null) {
            Visitor visitor = UserSearch.BY_ID.findFirst(visitorID);
            if(visitor != null && visitor.getInLibrary()) {
                Visit visit = LBMS.getCurrentVisits().remove(visitor.getVisitorID());
                visit.depart();
                LBMS.getTotalVisits().add(visit);
                long s = visit.getDuration().getSeconds();
                String duration = String.format("%02d:%02d:%02d", s / 3600, (s % 3600) / 60, (s % 60));
                return visitorID + "," + visit.getDepartureTime().format(SystemDateTime.TIME_FORMAT) + "," +
                        duration + ";";
            }
            return "invalid-id;";
        }
        return "invalid-id;";
    }

    /**
```

```
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        if(fields[1].equals("invalid-id;")) {
            return "\nVisitor " + visitorID + " is not in the library.";
        }
        else {
            return "\nVisitor " + visitorID + " has left the library at "
                    + fields[2] + " and was in the library for " + fields[3];
        }
    }
}
```

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;

    /**
     * Constructor for FindBorrowed class.
     * @param request: the request String for the command
     */
    public FindBorrowed(String request) {
        visitorID = Long.decode(request);
    }

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

        Visitor visitor = UserSearch.BY_ID.findFirst(visitorID);
```

```
        String s = "";
        s += visitor.getNumCheckedOut();
        final int[] id = {1};

        Book b;
        LBMS.getLastBookSearch().clear();
        for(Transaction t: visitor.getCheckedOutBooks().values()) {
            b = BookSearch.BY_ISBN.search(t.getIsbn()).get(0);
            LBMS.getLastBookSearch().add(b);
            s += "\n";
            s += id[0]++ + "," + t.getIsbn() + "," + b.getTitle() + "," + t.getDate();
        }

        return s + ";";
    }

    /**
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.replace(";", "").split("\n", 2);
        if (fields.length == 1) {
            if (fields[0].endsWith("0")) {
                return "\nVisitor " + visitorID + " has no borrowed books.";
            } else {
                return "\nVisitor " + visitorID + " is not valid.";
            }
        }
        else {
            return "\n" + fields[1];
        }
    }
}
```

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().getDate().format(SystemDateTime.DATE_FORMAT) + "," +
                SystemDateTime.getInstance().getTime().format(SystemDateTime.TIME_FORMAT) + ";";
    }

    /**
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        return "\nCurrent System Time: " + fields[1] + " " + fields[2];
    }
}
```

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;";
    }

    /**
     * Parses the response for standard output
     * @param response: the response string from execute
     * @return the output to be printed
     */
    public String parseResponse(String response) {
        if(response.equals("invalid-command;")) {
            return "Invalid Command.\n";
        }
        return null;
    }
}
```

LibrarySearch.java

```
package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.search.BookSearch;

import java.util.*;

/**
 * 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 Long isbn = null;

    /**
     * Constructor for a LibrarySearch command object.
     * @param request: the request string for a library search
     * @throws MissingParametersException: missing parameters
     */
    public LibrarySearch(String request) throws MissingParametersException {
        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(sort_order == null && (Arrays.asList(arguments).contains("title") ||
                    Arrays.asList(arguments).contains("publish-date") ||
                    Arrays.asList(arguments).contains("book-status"))) {
                    sort_order = arguments[arguments.length - 1];
                }
                if(arguments[index].startsWith("{")) {
                    authors = new ArrayList<>();
                    while(!arguments[index].endsWith("}")) {
                        authors.add(arguments[index++].replaceAll("[{}]", ""));
                    }
                    authors.add(arguments[index].replaceAll("[{}]", ""));
                }
                else if(!arguments[index].equals("*")) {
                    if(title == null && !arguments[0].equals("*")) {
                        title = (arguments[index]);
                    }
                    else if(isbn == null && arguments[index].matches("^\\d{13}$")) {
                        isbn = Long.parseLong(arguments[index]);
                    }
                    else if((publisher == null && sort_order == null && index == (arguments.length) - 1) ||
```

```
                    (publisher == null && sort_order != null && index == (arguments.length) - 2)) {
                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(sort_order != null && !sort_order.equals("title") && !sort_order.equals("publish-date") &&
            !sort_order.equals("book-status")) {
        return "invalid-sort-order;";
    }
    List<Book> matches;
    List<Book> antiMatches = new ArrayList<>();
    if(title != null) {
        matches = BookSearch.BY_TITLE.search(title);
    }
    else if(authors != null) {
        matches = BookSearch.BY_AUTHOR.search(authors.get(0));
    }
    else if(isbn != null) {
        matches = BookSearch.BY_ISBN.search(isbn);
    }
    else if(publisher != null) {
        matches = BookSearch.BY_PUBLISHER.search(publisher);
    }
    else {
        matches = new ArrayList<>();
    }
    for(Book b: matches) {
        if(title != null && !b.getTitle().contains(title)) {
            antiMatches.add(b);
        }
        if(authors != null) {
            for(String author: authors) {
                if(!b.hasAuthorPartial(author)) {
                    antiMatches.add(b);
                }
            }
        }
        if(isbn != null && b.getIsbn() != isbn) {
            antiMatches.add(b);
        }
        if(publisher != null && !b.getPublisher().equals(publisher)) {
            antiMatches.add(b);
        }
    }
```

```
        for(Book b: antiMatches) {
            matches.remove(b);
        }

        if(sort_order != null) {
            switch(sort_order) {
                case "title":
                    Collections.sort(matches, (Book b1, Book b2) -> b2.getTitle().compareTo(b1.getTitle()));
                    break;
                case "publish-date":
                    Collections.sort(matches, (Book b1, Book b2) -> b2.getPublishDate().compareTo(b1.getPublishDate()));
                    break;
                case "book-status":
                    Collections.sort(matches, (Book b1, Book b2) ->
                            ((Integer)b2.getCopiesAvailable()).compareTo(b1.getCopiesAvailable()));
                    break;
            }
        }
        LBMS.getLastBookSearch().clear();
        String matchesString = "";
        for(Book b: matches) {
            LBMS.getLastBookSearch().add(b);
            matchesString += "\n" + b.getCopiesAvailable() + "," + (LBMS.getLastBookSearch().indexOf(b) + 1) + "," +
                    b.toString() + ",";
        }
        if(matches.size() > 0) {
            matchesString = matchesString.substring(0, matchesString.length() - 1);
        }
        else {
            return "0;";
        }

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

    /**
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.replace(";", "").split("\n", 2);

        if (fields.length == 1) {
            if (fields[0].endsWith("0")) {
                return "No books match query.";
            } else {
                return response;
            }
        }
        else {
            return fields[1];
        }
    }
}
```

PayFine.java

```
package lbms.command;

import lbms.search.UserSearch;

import java.text.DecimalFormat;

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

    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.replaceAll(";$", "").split(",");
        visitorID = Long.parseLong(arguments[0]);
        amount = Double.parseDouble(arguments[1]);
    }

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

    /**
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
```

```
        if(fields[1].equals("invalid-visitor-id;")) {
            return "\nVisitor " + visitorID + " is not registered in the system.";
        }
        else if(fields[1].equals("invalid-amount")) {
            return "The specified amount, " + amount + " is not valid as it is negative or exceeds their balance, " +
                    UserSearch.BY_ID.findFirst(visitorID).getFines() + ".";
        }
        else {
            return "The fine has been successfully paid for. The remaining balance is " +
                    UserSearch.BY_ID.findFirst(visitorID).getFines() + ".";
        }
    }
}
```

RegisterVisitor.java

```
package lbms.command;

import lbms.LBMS;
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(",");
        try {
            visitor = new Visitor(arguments[0], arguments[1], arguments[2], Long.parseLong(arguments[3]));
        }
        catch(ArrayIndexOutOfBoundsException | NumberFormatException e) {
            if (arguments.length == 1 && arguments[0].equals("")) {
                throw new MissingParametersException("missing-parameters,first-name,last-name,address,phone-number");
            }
            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");
            }
            else {
```

```
            throw new MissingParametersException("missing-parameters,first-name,last-name,address,phone-number");
        }
    }
}

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

/**
 * Parses the response for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    String[] fields = response.split(",");
    if(fields[1].equals("duplicate;")) {
        return "This user already exists in the system.";
    }
    else {
        return String.format("\nNew visitor created on %s:\n\tName: %s\n\tAddress: %s\n\tPhone: %s\n\tVisitor " +
                    "ID: %d", fields[2].replace(";", ""), visitor.getName(), visitor.getAddress(),
                visitor.getPhoneNumber(), visitor.getVisitorID());
    }
}

/**
 * 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() == visitor.getPhoneNumber()) {
                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;
  }
}
```

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().reset();
            return "success;";
        }
        catch(Exception e) {
            return "failure;";
        }
    }

    /**
     * Parses the response for standard output.
     * @param response: the response string from execute
     * @return the output to be printed
     */
    @Override
    public String parseResponse(String response) {
        String[] fields = response.split(",");
        if(fields[1].equals("success;")) {
            return "\nSuccess, system clock has been reset";
        }
```

```
        else {
            return "\nFailure, system clock failed to reset";
        }
    }
}
```

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

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

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

    /**
     * Constructor for a Return command object.
     * @param request: the request input string
     */
    public Return(String request) {
        request = request.replaceAll(";$", "").replaceAll("\"", "");
        String[] split = request.split(",", 2);
        visitorID = Long.parseLong(split[0]);
        ids = Arrays.stream(split[1].split(",")).map(Integer::parseInt).collect(Collectors.toList());
    }

    /**
     * Executes the return command.
     * @return a response string or error message
     */
    @Override
    public String execute() {
        if(UserSearch.BY_ID.findFirst(visitorID) == null) {
            return "invalid-visitor-id;";
        }
        Visitor visitor = UserSearch.BY_ID.findFirst(visitorID);
        ArrayList<Integer> nonBooks = new ArrayList<>();
        for(Integer id : ids) {
            if(LBMS.getLastBookSearch().size() <= id) {
```

```
            try {
                Book b = LBMS.getLastBookSearch().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,";
        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().getDate().isAfter(t.getDueDate())) {
                output += LBMS.getLastBookSearch().indexOf(LBMS.getBooks().get(t.getIsbn())) + ",";
            }
        }
        return output.replaceAll(",$", ";");
    }

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

    return "success;";
}

/**
 * Parses the string for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    switch(response.replaceAll(";$", "") .split(",")[0]) {
        case "invalid-visitor-id":
            return "Invalid visitor ID entered.";
        case "invalid-book-id":
            return "Invalid book ID entered.";
        case "success":
            return "Book(s) successfully returned.";
```

```
            case "overdue":
                return "This book is overdue.";
            default:
                return "Unknown option/command.";
        }
    }
}
```

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("")) {
                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().getDate().format(SystemDateTime.DATE_FORMAT) + ",\n" +
generateReport(days);
    }
```

```
/**
 * Parses the response for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    String[] fields = response.split(",");
    return fields[fields.length - 1];
}

/**
 * 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().getDate().minusDays(days);
        LocalDate reportEndDate = SystemDateTime.getInstance().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));
  }
}
```

StoreSearch.java

```
package lbms.command;

import lbms.LBMS;
import lbms.models.Book;
import lbms.search.BookSearch;

import java.util.ArrayList;
import java.util.Arrays;
```

```
import java.util.Collections;
import java.util.List;

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

    private String title;
    private ArrayList<String> authors;
    private Long isbn = null;
    private String publisher = null;
    private String sortOrder = null;

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

                if(arguments[index].startsWith("{")) {
                    authors = new ArrayList<>();
                    while (!arguments[index].endsWith("}")) {
                        authors.add(arguments[index++].replaceAll("[{}]", ""));
                    }
                    authors.add(arguments[index].replaceAll("[{}]", ""));
                }
                else if(!arguments[index].equals("*")) {
                    if(title == null && !arguments[0].equals("*")) {
                        title = (arguments[index]);
                    }
                    else if(isbn == null && arguments[index].matches("^\\d{13}$")) {
                        isbn = Long.parseLong(arguments[index]);
                    }
                    else if((publisher == null && sortOrder == null && index == (arguments.length) - 1) ||
                            (publisher == null && sortOrder != null && index == (arguments.length) - 2)) {
                        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(sortOrder != null && !sortOrder.equals("title") && !sortOrder.equals("publish-date")) {
        return "invalid-sort-order";
    }
    List<Book> books = BookSearch.BY_TITLE.searchBookstoBuy(title);
    List<Book> remove = new ArrayList<>();
    if(authors != null) {
        for(Book b: books) {
            for(String author: authors) {
                if(!b.hasAuthorPartial(author)) {
                    remove.add(b);
                }
            }
        }
    }
    if(isbn != null) {
        for(Book b: books) {
            if(b.getIsbn() != isbn) {
                remove.add(b);
            }
        }
    }
    if(publisher != null) {
        for(Book b: books) {
            if(!b.getPublisher().contains(publisher)) {
                remove.add(b);
            }
        }
    }
    for(Book b: remove) {
        books.remove(b);
    }

    if(sortOrder != null && sortOrder.equals("title")) {
        Collections.sort(books, (Book b1, Book b2) -> b2.getTitle().compareTo(b1.getTitle()));
    }
    else if(sortOrder != null && sortOrder.equals("publish-date")) {
        Collections.sort(books, (Book b1, Book b2) -> b2.getPublishDate().compareTo(b1.getPublishDate()));
    }
    if(books.size() == 0) {
        return "0;";
    }
    else {
        int id = 1;
        String response = Integer.toString(books.size()) + "\n";
        LBMS.getLastBookSearch().clear();
        for(Book book : books) {
            LBMS.getLastBookSearch().add(book);
            response = response + id + "," + book.getIsbn() + "," + book.getTitle()
            + ",{";
```

```
            for(String author : book.getAuthors()) {
                response = response + author + ",";
            }
            response = response.replaceAll(",$", "},");
            response = response + book.dateFormat() + ",\n";
            id += 1;
        }
        response = response.substring(0, response.length() - 2);
        response += ";";
        return response;
    }
}

/**
 * Parses the response for standard output.
 * @param response: the response string from execute
 * @return the output to be printed
 */
@Override
public String parseResponse(String response) {
    String[] fields = response.replace(";", "").split("\n", 2);

    if (fields.length == 1) {
        if (fields[0].endsWith("0")) {
            return "No books match query.";
        } else {
            return response;
        }
    }
    else {
        return fields[1];
    }
}
}
```

Controllers Package

CommandController.java

```
package lbms.controllers;

import lbms.command.*;

/**
 * CommandController class interacts with the command package to execute commands.
 * @author Team B
 */
public class CommandController {

    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 static String processRequest(boolean SYSTEM_STATUS, String requestString) {
        String response = "";

        if(requestString.endsWith(";")) {
            String[] request = requestString.replace(";", "").split(",", 2);
            response = request[0] + ",";
            try {
                command = createCommand(SYSTEM_STATUS, request);
                response += command.execute();
            }
            catch(MissingParametersException e) {
                response += e.getMessage() + ";";
            }
            catch(Exception e) {
                response += "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;
    }

    /**
```

```
     * Creates a command based on the input request.
     * @param SYSTEM_STATUS: whether or not the system is operational
     * @param request: the input request to be processed
     * @return a Command object for the request
     */
    private static Command createCommand(boolean SYSTEM_STATUS, String[] request) throws Exception {
        switch (request[0]) {
            case "arrive":
                if(SYSTEM_STATUS) {
                    return new BeginVisit(request[1]);
                }
            case "borrow":
                if(SYSTEM_STATUS) {
                    return new Borrow(request[1]);
                }
                return new CloseLibrary();
            case "register":
                return new RegisterVisitor(request[1]);
            case "depart":
                return new EndVisit(request[1]);
            case "info":
                return new LibrarySearch(request[1]);
            case "borrowed":
                return new FindBorrowed(request[1]);
            case "return":
                return new Return(request[1]);
            case "pay":
                return new PayFine(request[1]);
            case "search":
                return new StoreSearch(request[1]);
            case "buy":
                return new BookPurchase(request[1]);
            case "advance":
                return new AdvanceTime(request[1]);
            case "datetime":
                return new GetDateTime();
            case "report":
                if(request.length == 1){
                    return new StatisticsReport("");
                }
                else {
                    return new StatisticsReport(request[1]);
                }
            case "reset":      // FOR TESTING
                return new ResetTime();
            default:
                return new Invalid();
        }
    }
}
```

ViewController.java

```
package lbms.controllers;
```

```
import lbms.views.State;

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

    private static State viewState;

    /**
     * Sets the state of the system.
     * @param state: the state to be set
     */
    public static void setState(State state) {
        viewState = state;
        viewState.flush();
        viewState.init();
        viewState.onEnter();
    }

    /**
     * Getter for the viewState variable.
     * @return the viewState
     */
    public static State getState() {
        return viewState;
    }

    /**
     * Changes the current state.
     * @param state: the state to be changed
     */
    public static void change(String state) {
        viewState.change(state);
    }
}
```

Models Package

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 long 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(long 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 title;
}

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

/**
 * Getter for the authors.
 * @return the list of authors of the book
 */
public ArrayList<String> getAuthors() {
    return 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.contains(name))
            .collect(Collectors.toList()).isEmpty();
}

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

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

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

/**
 * Getter for the published date.
 * @return the publishing date for the book
```

```
 */
public Calendar getPublishDate() {
    return publishDate;
}

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

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

/**
 * Returns a book.
 */
public void returnBook() {
    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(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() {
        purchaseDate = SystemDateTime.getInstance().getDate();
        numberOfCopies++;
    }
}
```

SystemDateTime.java

```
package lbms.models;

import java.io.Serializable;
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 implements Serializable {

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

    /** 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");

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

```
    }
}

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

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

/**
 * Sets the instance of the system date time.
 * @param inst: the instance to be set
 */
public static void setInstance(SystemDateTime inst) {
    instance = inst;
}

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

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

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

/**
 * Creates a string of the system date time.
 * @return string representation of the system date time
```

```
     */
    public String toString() {
        return time.format(DATETIME_FORMAT);
    }

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

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

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

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

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 long 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(long isbn, long visitorId) {
    this.isbn = isbn;
    this.visitorId = visitorId;
    this.date = SystemDateTime.getInstance().getDate();
    this.dueDate = date.plusDays(7);
}

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

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

/**
 * Getter for the fine.
 * @return the fine due on the book
 */
double getFine() {
    int days = Period.between(dueDate, SystemDateTime.getInstance().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() {
        closeDate = SystemDateTime.getInstance().getDate();
    }

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

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

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().getDateTime();
        this.timeOfDeparture = null;
        this.duration = null;
        this.visitor.switchInLibrary(true);
    }

    /**
```

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

    /**
     * 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 dateTime.toLocalDate();
    }

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

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

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

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 address;
    private long phoneNumber;
    private long visitorID;
    private HashMap<Long, Transaction> checkedOutBooks;
    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 first name of the visitor
     * @param lastName: the last name of the visitor
     * @param address: the address of the visitor
     * @param phoneNumber: the visitor's phone number
     */
    public Visitor(String firstName, String lastName, String address, long phoneNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
        this.phoneNumber = phoneNumber;
        this.visitorID = LBMS.getVisitors().size() + 1;
        this.checkedOutBooks = new HashMap<>(MAX_BOOKS);
        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 firstName + " " + lastName;
    }

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

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

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

/**
 * Getter for the number of books the visitor has checked out.
 * @return the number of checked out books
 */
public int getNumCheckedOut() {
    return checkedOutBooks.size();
}

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

/**
 * 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 && !(totalFines + currentFines > payedFines);
}

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

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

```
        checkedOutBooks.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 inLibrary;
    }

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

    /**
     * Determines the fines the visitor owes.
     * @return the amount of fines due
     */
    public double getFines() {
        double fines = 0;
        for(Long l: checkedOutBooks.keySet()) {
            fines += 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) {
        payedFines += amount;
    }

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

Search Package

Search.java

```
package lbms.search;

import java.util.List;

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

    /**
     * Finds objects that fit the search criteria
     * @param s: the string to search for
     * @return a list of objects that match
     */
    List<T> search(String s);

    /**
     * Finds the first instance of l.
     * @param l: the long to be searched for
     * @return the first instance of l
     */
    default T findFirst(Long l) {
        return findFirst(Long.toString(l));
    }

    /**
     * Finds the first instance of s.
     * @param s: the string to be searched for
     * @return the first instance of s
     */
    default T findFirst(String s) {
        List<T> results = search(s);
        return results.isEmpty() ? null : results.get(0);
    }

}
```

BookSearch.java

```
package lbms.search;

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

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;
```

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

    /**
     * Searches the books by l.
     * @param l: the isbn of the book
     * @return a list of books that match
     */
    public List<Book> search(long l) {
        return search(Long.toString(l));
    }

    /**
     * Searches the books by author, isbn, or title.
     * @param s: the string to search for
     * @return a list of books that match
     */
    @Override
    public List<Book> search(String s) {
        switch(this) {
            case BY_AUTHOR:
                return find(book -> book.hasAuthorPartial(s));
            case BY_ISBN:
                return find(book -> Long.toString(book.getIsbn()).contains(s));
            case BY_TITLE:
                return find(book -> book.getTitle().contains(s));
            case BY_PUBLISHER:
                return find(book -> book.getPublisher().contains(s));
        }
        return new ArrayList<>();
    }

    /**
     * Finds the books.
     * @param condition: the condition that must be true
     * @return a list of books that match
     */
    private List<Book> find(Predicate<? super Book> condition) {
        return LBMS.getBooks().values().parallelStream().filter(condition).collect(Collectors.toList());
    }

    /**
     * Finds the books from the book store.
     * @param s: the string used for searching
     * @return a list of books that match the search
     */
    public List<Book> searchBookstoBuy(String s) {
        switch(this) {
```

```
            case BY_AUTHOR:
                return findBooksToBuy(book -> book.hasAuthorPartial(s));
            case BY_ISBN:
                return findBooksToBuy(book -> Long.toString(book.getIsbn()).contains(s));
            case BY_TITLE:
                return findBooksToBuy(book -> book.getTitle().contains(s));
            case BY_PUBLISHER:
                return findBooksToBuy(book -> book.getPublisher().contains(s));
        }
        return new ArrayList<>();
    }

    /**
     * Finds the books.
     * @param condition: the condition that must be true
     * @return a list of books that match
     */
    private List<Book> findBooksToBuy(Predicate<? super Book> condition) {
        return LBMS.getBooksToBuy().parallelStream().filter(condition).collect(Collectors.toList());
    }
}
```

UserSearch.java

```
package lbms.search;

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

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

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

    /**
     * Searches the users by id.
     * @param l: the id of the users
     * @return the list of visitors that match
     */
    public List<Visitor> search(long l) {
        return search(Long.toString(l));
    }

    /**
     * Searches the visitors by id, name, or address.
```

```
     * @param s: the string to search for
     * @return a list of visitors that match
     */
    @Override
    public List<Visitor> search(String s) {
        switch(this) {
            case BY_ID:
                return find(visitor -> Long.toString(visitor.getVisitorID()).equals(s));
            case BY_NAME:
                return find(visitor -> visitor.getName().equals(s));
            case BY_ADDRESS:
                return find(visitor -> visitor.getAddress().equals(s));
            case BY_PHONE:
                return find(visitor -> visitor.getPhoneNumber() == Long.parseLong(s));
        }
        return new ArrayList<>();
    }

    /**
     * Finds the first visitor.
     * @param l: the id to be searched for
     * @return a visitor with the given id
     */
    @Override
    public Visitor findFirst(Long l) {
        if(l.toString().length() <= 10) {
            return LBMS.getVisitors().get(l);
        }
        return findFirst(Long.toString(l));
    }

    /**
     * Finds the list of visitors that meet the condition.
     * @param condition: the condition that must be true
     * @return the list of visitors that match
     */
    private List<Visitor> find(Predicate<? super Visitor> condition) {
        return LBMS.getVisitors().values().parallelStream().filter(condition).collect(Collectors.toList());
    }
}
```

Views Package

State.java

```
package lbms.views;

/**
 * Abstract representation of a views.
 *
 * Note: This is not an interface, as to ensure correct access modifiers.
 */
public interface State {

    /**
     * Updates the views. Should only be called internally.
     */
    void init();

    /**
     * Called every time the views is entered.
     */
    void onEnter();

    /**
     * Handle a command passed to the views
     * @param state: the command to handle
     */
    void change(String state);

    /**
     * Used to flush the console.
     */
    default void flush() {
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }

}
```

AdvanceViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * Advance view state for the views package.
 * @author Team B
 */
public class AdvanceViewState implements State {
```

```
    private boolean SYSTEM_STATUS;
    private int days;
    private int hours;

    /**
     * Constructor for an AdvanceViewState.
     * @param SYSTEM_STATUS: the current status of the system
     */
    AdvanceViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes AdvanceViewState.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nHow many days would you like to advance the clock?");
        days = scanner.nextInt();
        System.out.println("How many hours would you like to advance the clock?");
        hours = scanner.nextInt();
    }

    /**
     * Processes the command for advancing the time.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"advance," + days + "," + hours + ";");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        } catch (Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new ClockViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

BeginVisitViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;
```

```
import java.util.Scanner;

/**
 * BeginVisitViewState class that processes the begin visit command.
 * @author Team B
 */
public class BeginVisitViewState implements State {

    private boolean SYSTEM_STATUS;
    private long visitorID;

    /**
     * Constructor for the BeginVisitViewState.
     * @param SYSTEM_STATUS: the initial status of the system
     */
    BeginVisitViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the begin visit view state.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("\nWhat is the ID of the visitor entering the library? ");
        visitorID = scanner.nextLong();
    }

    /**
     * Processes the command string for begin visit.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"arrive," + visitorID + ";");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        } catch (Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

BookSearchMenuViewState.java

```
package lbms.views;

import lbms.controllers.ViewController;

/**
 * Book Search Menu view for views package.
 * @author Team B
 */
public class BookSearchMenuViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for BookSearchMenuViewState.
     * @param SYSTEM_STATUS: the status of the system
     */
    BookSearchMenuViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the state.
     */
    @Override
    public void init() {
        System.out.println("\nPlease select a command:");
        System.out.println("library    Search for a book in the library");
        System.out.println("store      Search for a book in the store");
        System.out.println("return     Return to main menu");
    }

    /**
     * No operation from this method.
     */
    @Override
    public void onEnter() {}

    /**
     * Method to change states.
     * @param state: the command to handle
     */
    public void change(String state) {
        switch(state) {
            case "library":
                ViewController.setState(new LibrarySearchViewState(SYSTEM_STATUS));
                break;
            case "store":
                ViewController.setState(new StoreSearchViewState(SYSTEM_STATUS));
                break;
            case "return":
                ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
                break;
            default:
```

```
            System.out.println("Command not found\n");
            this.init();
        }
    }
}
```

BooksListViewState.java

```
package lbms.views;

import lbms.LBMS;
import lbms.controllers.ViewController;
import lbms.models.Book;

/**
 * BooksListViewState class for views package.
 * @author Team B
 */
public class BooksListViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for BooksListViewState.
     * @param SYSTEM_STATUS: the status of the system
     */
    BooksListViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the state.
     */
    @Override
    public void init() {
        System.out.println();

        if(LBMS.getBooks().isEmpty()) {
            System.out.println("No books are owned by the library.");
        }
        else {
            for(Book book : LBMS.getBooks().values()) {
                System.out.println(book.toString());
            }
        }

        ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     */
    @Override
    public void onEnter() {}
```

```
    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

BooksMenuViewState.java

```
package lbms.views;

import lbms.controllers.ViewController;

/**
 * BooksMenuViewState class for viewing books in the system.
 * @author Team B
 */
public class BooksMenuViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for a BooksMenuViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    BooksMenuViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initialized the BooksMenuViewState.
     */
    @Override
    public void init() {
        System.out.println("\nPlease select a command:");
        System.out.println("search      Search for a book");

        if(SYSTEM_STATUS) {
            System.out.println("checkout    Borrow a book");
        }

        System.out.println("checkin     Return a book");
        System.out.println("list        Show all available books");
        System.out.println("return      Return to main menu");
    }

    /**
     * No operation from this method.
     */
    @Override
    public void onEnter() { }

    /**
     * Changes the state.
```

```
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {
        switch(state) {
            case "search":
                ViewController.setState(new BookSearchMenuViewState(SYSTEM_STATUS));
                break;
            case "list":
                ViewController.setState(new BooksListViewState(SYSTEM_STATUS));
                break;
            case "checkin":
                ViewController.setState(new ReturnBookViewState(SYSTEM_STATUS));
                break;
            case "return":
                ViewController.setState(new DefaultViewState(SYSTEM_STATUS));
                break;
            case "checkout":
            case "borrow":
                if(SYSTEM_STATUS) {
                    ViewController.setState(new BorrowBookViewState(true));
                    break;
                }
            default:
                System.out.println("Command not found\n");
                this.init();
        }
    }
}
```

BorrowBookViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * BorrowBookViewState class for views package.
 * @author Team B
 */
public class BorrowBookViewState implements State {

    private boolean SYSTEM_STATUS;
    private long visitorID;
    private String books = "";

    /**
     * Constructor for a BorrowBookViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    BorrowBookViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
```

```
    }

    /**
     * Initializes the state.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nWhat is the ID of the visitor borrowing the book? ");
        visitorID = scanner.nextLong();
        String response;
        do {
            System.out.println("What is the id of the book they are borrowing?");
            books += "," + scanner.next();
            System.out.println("Is the visitor borrowing another book?");
            response = scanner.next();
        } while(response.toLowerCase().equals("yes") || response.toLowerCase().equals("y"));
    }

    /**
     * Method handles what happens after the state is initialized.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS, "borrow," + visitorID + books
                + ";");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

ClockViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

/**
 * ClockViewState class processes the clock command.
 * @author Team B
```

```
 */
public class ClockViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for a ClockViewState.
     * @param SYSTEM_STATUS: the status of the system
     */
    ClockViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the ClockViewState.
     */
    @Override
    public void init() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"datetime;");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        System.out.println("clock      View system time");
        System.out.println("advance    Fast-forward clock");
        System.out.println("reset      Reset the clock to current time");
        System.out.println("return     Return to main menu");
    }

    /**
     * No operation from this method.
     */
    @Override
    public void onEnter() {}

    /**
     * Changes the state.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {
        switch(state) {
            case "clock":
                this.init();
                break;
            case "advance":
                ViewController.setState(new AdvanceViewState(SYSTEM_STATUS));
                break;
            case "reset":
                ViewController.setState(new ResetViewState(SYSTEM_STATUS));
                break;
            case "return":
```

```
            ViewController.setState(new SystemViewState(SYSTEM_STATUS));
            break;
        default:
            System.out.println("Command not found\n");
            this.init();
      }
   }
}
```

DefaultViewState.java

```
package lbms.views;

import lbms.controllers.ViewController;

/**
 * This is the default views which is entered when the system starts.
 * @author Team B
 */
public class DefaultViewState implements State {

   private boolean SYSTEM_STATUS;

   /**
    * Constructor for a DefaultViewState.
    * @param SYSTEM_STATUS: the status of the system
    */
   public DefaultViewState(boolean SYSTEM_STATUS) {
      this.SYSTEM_STATUS = SYSTEM_STATUS;
   }

   /**
    * Prompts a user whether to views books or users, or exit.
    */
   @Override
   public void init() {
      System.out.println("\nWelcome to the Library Book Management System!");

      if(!SYSTEM_STATUS) {
         System.out.println("We are currently closed but here you can still access a few commands.");
      }

      System.out.println("\nPlease select a command: ");
      System.out.println("books)     View books");
      System.out.println("users)     View users");
      System.out.println("system)    Edit system");
      System.out.println("exit)      Exit system");
   }

   /**
    * No operation from this method.
    */
   @Override
   public void onEnter() {}
```

```
    /**
     * Changes the state.
     * @param state: the command to handle
     */
    public void change(String state) {
        switch(state) {
            case "books":
                ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
                break;
            case "users":
                ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
                break;
            case "system":
                ViewController.setState(new SystemViewState(SYSTEM_STATUS));
                break;
            default:
                System.out.println("Command not found\n\n");
                this.init();
                break;
        }
    }
}
```

EndVisitViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * EndVisitViewState class for views package.
 * @author Team B
 */
public class EndVisitViewState implements State {

    private boolean SYSTEM_STATUS;
    private long visitorID;

    /**
     * Constructor for an EndVisitViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    EndVisitViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the view.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("\nWhat is the ID of the visitor exiting the library? ");
        visitorID = scanner.nextLong();
    }

    /**
     * Processes the command.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"depart," + visitorID + ";");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        } catch (Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

FindBorrowedViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * FindBorrowedViewState class.
 * @author Team B
 */
public class FindBorrowedViewState implements State {

    private boolean SYSTEM_STATUS;
    private long visitorID;

    /**
     * Constructor for FindBorrowedViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    FindBorrowedViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
```

```
     * Initializes the view.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nWhat is the ID of the visitor you are querying?");
        visitorID = scanner.nextLong();
    }

    /**
     * Processes the command.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"borrowed," + visitorID + ";");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        } catch (Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

LibrarySearchViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * Interacts with user to orchestrate a library search.
 * @author Team B
 */
public class LibrarySearchViewState implements State{

    private boolean SYSTEM_STATUS;
    private String commandString = "info";
    private String[] prompts = {
            "\nPlease enter the title of the book to search for:",
            "\nPlease enter the author(s) of the book to search for (comma separated):",
            "\nPlease enter the isbn of the book to search for:",
            "\nPlease enter the publisher of the book to search for:",
```

```
        "\nPlease enter the sort-order for the resulting books:",
};

/**
 * Constructor for an LibrarySearchViewState.
 * @param SYSTEM_STATUS the current status of the system
 */
LibrarySearchViewState(boolean SYSTEM_STATUS) {
    this.SYSTEM_STATUS = SYSTEM_STATUS;
}

/**
 * Produces the command string based on user input.
 */
@Override
public void init() {
    System.out.println("\nYou are now searching the library");
    System.out.println("(Enter \"*\" to skip any step)");

    Scanner scanner = new Scanner(System.in);
    String input;
    String optionalArgumentPrompt = "(Press enter to search only with what you've input so far)";
    for(String prompt : prompts) {
        System.out.println(prompt);
        if(!prompt.equals(prompts[0]) && !prompt.equals(prompts[1])) {
            System.out.println(optionalArgumentPrompt);
        }

        input = scanner.nextLine();
        if(input.equals("")) {
            break;
        }
        else {
            if(prompt.equals(prompts[1])) {
                commandString += ",{" + input + "}";
            }
            else {
                commandString += "," + input;
            }
        }
    }
}

/**
 * Processes the command for searching the library.
 */
@Override
public void onEnter() {
    String response = CommandController.processRequest(this.SYSTEM_STATUS, commandString + ";");

    try {
        System.out.println(CommandController.getCommand().parseResponse(response));
    }
    catch(Exception e){
        System.out.println(response);
    }
```

```
        ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

PurchaseBookViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * PurchaseBookViewState class for views package.
 * @author Team B
 */
public class PurchaseBookViewState implements State {

    private boolean SYSTEM_STATUS;
    private int quantity;
    private String ids = "";

    /**
     * Constructor for an PurchaseBookViewState.
     * @param SYSTEM_STATUS: the current status of the system
     */
    PurchaseBookViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the PurchaseBook State
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);

        System.out.println("\nWhat quantity of these books would you like to purchase?");
        quantity = scanner.nextInt();
        String response;
        do {
            System.out.println("\nPlease enter the ID of the book to purchase.");
            ids += "," + scanner.next();
            System.out.println("\nAre you buying another book?");
            response = scanner.next();
        } while(response.toLowerCase().equals("yes") || response.toLowerCase().equals("y"));
```

```
    }

    /**
     * Processes the command for purchasing a book.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"buy," + quantity + ids + ";");

        try {
            System.out.println("\n" + CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

RegisterViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * This views handles registering a new user.
 * @author Team B
 */
public class RegisterViewState implements State {

    private boolean SYSTEM_STATUS;
    private String firstName;
    private String lastName;
    private String address;
    private long phone;

    /**
     * Constructor for a RegisterViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    RegisterViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }
```

```
    /**
     * Prompts the user to verify the entered information.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nRegister a new user.");
        System.out.print("First Name: ");
        firstName = scanner.nextLine();
        System.out.print("Last Name: ");
        lastName = scanner.nextLine();
        System.out.print("Address: ");
        address = scanner.nextLine();
        System.out.print("Phone Number: ");
        phone = Long.parseLong(scanner.nextLine().replaceAll("[\\D]", ""));
    }

    /**
     * Get information from the user to register a new user
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"register," + firstName + ","
                + lastName + "," + address + "," + phone + ";");
        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

ReportViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * Report View for fiew package
 */
```

```
public class ReportViewState implements State {

    private boolean SYSTEM_STATUS;
    private Integer days = null;

    /**
     * Constructor for a SystemViewState.
     * @param SYSTEM_STATUS: the status of the system
     */
    ReportViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Prompts a user whether to views books or users, or exit
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nHow many days would you like a report for? Type \"all\" to get all statistics.");
        String input = scanner.nextLine();

        try {
            days = Integer.parseInt(input);
        }
        catch(Exception e){
            days = null;
        }
    }

    /**
     * Method handles the state after initialization.
     */
    @Override
    public void onEnter() {
        String response;
        if(days == null) {
            response = CommandController.processRequest(this.SYSTEM_STATUS, "report;");
            System.out.println("\nSystem report:");
        }
        else {
            response = CommandController.processRequest(this.SYSTEM_STATUS, "report," + days + ";");
            System.out.println("\nSystem report for " + days + " days:");
        }

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new SystemViewState(SYSTEM_STATUS));
    }

    /**
```

```
     * No operation for this method.
     * @param state: the command to handle
     */
    public void change(String state) {}
}
```

ResetViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

/**
 * ResetViewState class.
 * @author Team B
 */
public class ResetViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for a ResetViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    ResetViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the view.
     */
    @Override
    public void init() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS,"reset;");

        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        }
        catch(Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new ClockViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     */
    @Override
    public void onEnter() {}

    /**
     * No operation from this method.
     * @param state: the command to handle
```

```
     */
    @Override
    public void change(String state) {}
}
```

ReturnBookViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * ReturnBookViewState class for views package.
 * @author Team B
 */
public class ReturnBookViewState implements State {

    private boolean SYSTEM_STATUS;
    private long visitorID;
    private String books = "";

    /**
     * Constructor for FindBorrowedViewState object.
     * @param SYSTEM_STATUS: the status of the system
     */
    ReturnBookViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the view.
     */
    @Override
    public void init() {
        Scanner scanner = new Scanner(System.in);
        System.out.println("\nWhat is the ID of the visitor returning the book? ");
        visitorID = scanner.nextLong();
        String response;
        do {
            System.out.println("What is the id of the book they are returning?");
            books += "," + scanner.next();
            System.out.println("Is the visitor returning another book?");
            response = scanner.next();
        } while(response.toLowerCase().equals("yes") || response.toLowerCase().equals("y"));
    }

    /**
     * Processes the command.
     */
    @Override
    public void onEnter() {
        String response = CommandController.processRequest(this.SYSTEM_STATUS, "return," + visitorID + books + ";");
```

```
        try {
            System.out.println(CommandController.getCommand().parseResponse(response));
        } catch (Exception e) {
            System.out.println(response);
        }

        ViewController.setState(new BooksMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation from this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

StoreSearchViewState.java

```
package lbms.views;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;

import java.util.Scanner;

/**
 * Interacts with user to orchestrate a book store search.
 * @author Team B
 */
public class StoreSearchViewState implements State {

    private boolean SYSTEM_STATUS;

    private String commandString = "search";
    private String[] prompts = {
            "\nPlease enter the title of the book to search for:",
            "\nPlease enter the author(s) of the book to search for (comma separated):",
            "\nPlease enter the isbn of the book to search for:",
            "\nPlease enter the publisher of the book to search for:",
            "\nPlease enter the sort-order for the resulting books:",
    };

    /**
     * Constructor for an StoreSearchViewState.
     * @param SYSTEM_STATUS the current status of the system
     */
    StoreSearchViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Produces the command string based on user input.
     */
```

```
@Override
public void init() {
    System.out.println("\nYou are now searching the store.");
    System.out.println("(Enter \"*\" to skip any step)");
    Scanner scanner = new Scanner(System.in);
    String input;
    String optionalArgumentPrompt = "(Press enter to search only with what you've input so far)";
    for(String prompt : prompts) {
        System.out.println(prompt);
        if(!prompt.equals(prompts[0])) {
            System.out.println(optionalArgumentPrompt);
        }

        input = scanner.nextLine();
        if(input.equals("")) {
            break;
        }
        else {
            if(prompt.equals(prompts[1])) {
                commandString += ",{" + input + "}";
            }
            else {
                commandString += "," + input;
            }
        }
    }
}

/**
 * Processes the command for searching the bookstore.
 */
@Override
public void onEnter() {
    String response = CommandController.processRequest(this.SYSTEM_STATUS, commandString + ";");

    try {
        System.out.println(CommandController.getCommand().parseResponse(response));
    } catch (Exception e) {
        System.out.println(response);
    }

    displayMenu();
}

/**
 * Displays the menu for this state.
 */
private void displayMenu() {
    System.out.println("\nPlease select a command:");
    System.out.println("purchase    Buy a book for the library from these search results");
    System.out.println("search      Search the store again");
    System.out.println("return      Return to main menu");
}

/**
 * Changes the state from this state.
```

```
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {
        switch(state) {
            case "purchase":
                ViewController.setState(new PurchaseBookViewState(SYSTEM_STATUS));
                break;
            case "search":
                ViewController.setState(new StoreSearchViewState(SYSTEM_STATUS));
                break;
            case "return":
                ViewController.setState(new BookSearchMenuViewState(SYSTEM_STATUS));
                break;
            default:
                System.out.println("Command not found\n");
                this.displayMenu();
                break;
        }
    }
}
```

SystemViewState.java

```
package lbms.views;

import lbms.controllers.ViewController;

/**
 * SystemViewState class.
 * @author Team B
 */
public class SystemViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for a SystemViewState.
     * @param SYSTEM_STATUS: the status of the system
     */
    SystemViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Prompts a user whether to views books or users, or exit
     */
    @Override
    public void init() {
        System.out.println("\nPlease select a command: ");
        System.out.println("clock     View system time");
        System.out.println("report    View system statistics");
        System.out.println("return    Return to main menu");
    }
```

```
    /**
     * No operation from this method.
     */
    @Override
    public void onEnter() {}

    /**
     * Changes the state of the system.
     * @param state: the command to handle
     */
    public void change(String state) {
        switch(state) {
            case "clock":
                ViewController.setState(new ClockViewState(SYSTEM_STATUS));
                break;
            case "report":
                ViewController.setState(new ReportViewState(SYSTEM_STATUS));
                break;
            case "return":
                ViewController.setState(new DefaultViewState(SYSTEM_STATUS));
                break;
            default:
                System.out.println("Command not found\n");
                this.init();
                break;
        }
    }
}
```

UserListViewState.java

```
package lbms.views;

import lbms.LBMS;
import lbms.controllers.ViewController;
import lbms.models.Visitor;

/**
 * UserListViewState class for views package.
 * @author Team B
 */
public class UserListViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for UserListViewState class.
     * @param SYSTEM_STATUS: the status of the system
     */
    UserListViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
     * Initializes the state.
```

```
     */
    @Override
    public void init() {
        System.out.println();

        if (LBMS.getVisitors().isEmpty()) {
            System.out.println("No users are registered in the system.");
        }
        else {
            for(Visitor visitor : LBMS.getVisitors().values()) {
                System.out.println(String.format("Visitor ID: %d\n\tName: %s\n\tAddress: %s\n\tPhone: %s\n",
                        visitor.getVisitorID(), visitor.getName(), visitor.getAddress(), visitor.getPhoneNumber()));
            }
        }

        ViewController.setState(new UserMenuViewState(SYSTEM_STATUS));
    }

    /**
     * No operation for this method.
     */
    @Override
    public void onEnter() {}

    /**
     * No operation for this method.
     * @param state: the command to handle
     */
    @Override
    public void change(String state) {}
}
```

UserMenuViewState.java

```
package lbms.views;

import lbms.controllers.ViewController;

/**
 * UserMenuViewState class.
 * @author Team B
 */
public class UserMenuViewState implements State {

    private boolean SYSTEM_STATUS;

    /**
     * Constructor for a UserMenuViewState object.
     * @param SYSTEM_STATUS: boolean status of the system
     */
    UserMenuViewState(boolean SYSTEM_STATUS) {
        this.SYSTEM_STATUS = SYSTEM_STATUS;
    }

    /**
```

```
 * Prompts a user to either search or register a user
 */
@Override
public void init() {
    System.out.println("\nPlease select a command:");

    if(SYSTEM_STATUS) {
        System.out.println("enter library)    Allow a user to enter the library");
        System.out.println("exit library)     Have a user leave the library");
    }

    System.out.println("register)         Register a new user");
    System.out.println("list)             List all the users in the system");
    System.out.println("borrowed)         Find the books a user has borrowed");
    System.out.println("return)           Return to main menu");
}

/**
 * No operation from this method.
 */
@Override
public void onEnter() { }

/**
 * Changes the state of the system.
 * @param state: the command to handle
 */
@Override
public void change(String state) {
    switch (state) {
        case "register":
            ViewController.setState(new RegisterViewState(SYSTEM_STATUS));
            break;
        case "list":
            ViewController.setState(new UserListViewState(SYSTEM_STATUS));
            break;
        case "borrowed":
            ViewController.setState(new FindBorrowedViewState(SYSTEM_STATUS));
            break;
        case "return":
            ViewController.setState(new DefaultViewState(SYSTEM_STATUS));
            break;
        case "enter library":
        case "enter":
            if(SYSTEM_STATUS) {
                ViewController.setState(new BeginVisitViewState(true));
                break;
            }
        case "exit library":
        case "leave":
            if(SYSTEM_STATUS) {
                ViewController.setState(new EndVisitViewState(true));
                break;
            }
        default:
            System.out.println("Command not found\n");
```

```
            this.init();
            break;
        }
    }
}
```

Main Package

LBMS.java

```
package lbms;

import lbms.controllers.CommandController;
import lbms.controllers.ViewController;
import lbms.models.*;
import lbms.views.DefaultViewState;

import java.io.*;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.LocalTime;
import java.util.*;

/**
 * Main class to run the Library Book Management System.
 * @author Team B
 */
public class LBMS {

    private final static LocalTime OPEN_TIME = LocalTime.of(8, 0);
    private final static LocalTime CLOSE_TIME = LocalTime.of(19, 0);

    private static LBMS instance;
    private static HashMap<Long, Book> books = new HashMap<>();
    private static ArrayList<Book> lastBookSearch = new ArrayList<>();
    private static ArrayList<Book> booksToBuy = new ArrayList<>();
    private static HashMap<Long, Visitor> visitors = new HashMap<>();
    private static ArrayList<Visit> totalVisits = new ArrayList<>();
    private static ArrayList<Transaction> transactions = new ArrayList<>();
    private static HashMap<Long, Visit> currentVisits = new HashMap<>();

    /**
     * Program entry point. Handle command line arguments and start.
     * @param args: the program arguments
     */
    public static void main(String[] args) {
        boolean console;
        try {
            console = Boolean.parseBoolean(args[0]);
        }
        catch(ArrayIndexOutOfBoundsException e) {
            console = true;
        }
        new LBMS(console);
    }

    /**
     * Handles user input for the LBMS system.
     */
    public LBMS(boolean console) {
```

```
SystemInit();
Scanner s = new Scanner(System.in);
int initial = 0;
if(console) {
   while(true) {
      // Check if library is open
      if(SystemDateTime.getInstance().getTime().isAfter(OPEN_TIME) &&
            SystemDateTime.getInstance().getTime().isBefore(CLOSE_TIME)) {
         // Check if library just opened or system start
         if(initial == 0 || initial == 1) {
            ViewController.setState(new DefaultViewState(true));
            initial = 2;
         }
      }
      else {
         // Check if library just closed or system start
         if(initial == 0 || initial == 2) {
            SystemClose();
            ViewController.setState(new DefaultViewState(false));
            initial = 1;
         }
      }

      System.out.print("> ");
      String input = s.nextLine();
      if(input.matches("(?i)exit|quit")) {
         break;
      }
      ViewController.change(input);
   }
}
else {
   String input;
   do {
      System.out.print("> ");
      input = s.nextLine();
      if(SystemDateTime.getInstance().getTime().isAfter(OPEN_TIME) &&
            SystemDateTime.getInstance().getTime().isBefore(CLOSE_TIME)) {
         // Check if library just opened or system start
         if(initial == 0 || initial == 1) {
            initial = 2;
         }
         System.out.println(CommandController.processRequest(true, input));
      }
      else {
         // Check if library just closed or system start
         if(initial == 0 || initial == 2) {
            SystemClose();
            initial = 1;
         }
         System.out.println(CommandController.processRequest(false, input));
      }
   } while(!input.matches("(?i)exit|quit"));
}
s.close();
SystemClose();
```

```
}

/**
 * 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;
        long isbn;
        int pageCount, i;
        Calendar publishDate = null;

        while(s.hasNextLine()) {
            i = 1;
            line = s.nextLine();
            parts = line.split(",");
            isbn = Long.parseLong(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) + ",";
        }
    }

    if(parts[parts.length-2].length() == 10) {
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
        try {
            Date date = format.parse(parts[parts.length - 2]);
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(date);
            publishDate = calendar;
        }
        catch(ParseException e) {
            e.printStackTrace();
        }
    }
    else if(parts[parts.length-2].length() == 7) {
        SimpleDateFormat format = new SimpleDateFormat("yyyy-MM");
        try {
            Date date = format.parse(parts[parts.length - 2]);
            Calendar calendar = Calendar.getInstance();
            calendar.setTime(date);
            publishDate = calendar;
        }
        catch(ParseException e) {
            e.printStackTrace();
        }
    }
    else if(parts[parts.length-2].length() == 4) {
        SimpleDateFormat format = new SimpleDateFormat("yyyy");
        try {
            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;
}

/**
 * Initializes the system.
 */
private void SystemInit() {
    instance = this;

    // Deserialize the data.
    try {
        FileInputStream f = new FileInputStream("data.ser");
        ObjectInputStream in = new ObjectInputStream(f);
        books = (HashMap<Long, Book>)in.readObject();
        booksToBuy = (ArrayList<Book>)in.readObject();
        visitors = (HashMap<Long, Visitor>)in.readObject();
        totalVisits = (ArrayList<Visit>) in.readObject();
        transactions = (ArrayList<Transaction>)in.readObject();
        SystemDateTime.setInstance((SystemDateTime) in.readObject());
    }
    catch(ClassNotFoundException | IOException e) {
        books = new HashMap<>();
        booksToBuy = makeBooks();
        visitors = new HashMap<>();
        totalVisits = new ArrayList<>();
        transactions = new ArrayList<>();
    }
    currentVisits = new HashMap<>();
    SystemDateTime systemDateTime = SystemDateTime.getInstance();
    systemDateTime.start();
}

/**
 * Serializes the data in the system for future startup.
 */
private void SystemClose() {
    SystemDateTime.getInstance().stopClock();

    // Departs all the visitors when the library closes.
    for(Visit visit: currentVisits.values()) {
        CommandController.processRequest(false, "depart," +
                visit.getVisitor().getVisitorID() + ";");
    }

    // 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(totalVisits);
            out.writeObject(transactions);
            out.writeObject(SystemDateTime.getInstance());
            out.close();
            f.close();
        }
        catch(IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
    }

    /**
     * Getter for the hash map of books
     * @return the books
     */
    public static HashMap<Long, Book> getBooks() {
        return books;
    }

    /**
     * Getter for the last set of books from a search.
     * @return the last books returned from a search
     */
    public static ArrayList<Book> getLastBookSearch() {
        return lastBookSearch;
    }

    /**
     * Getter for the books to be purchased by the library.
     * @return the array list of books that can be purchased
     */
    public static ArrayList<Book> getBooksToBuy() {
        return booksToBuy;
    }

    /**
     * Getter for the visitors.
     * @return an array list of visitors of the library
     */
    public static HashMap<Long, Visitor> getVisitors() {
        return visitors;
    }

    /**
     * Getter for the visits made by visitors.
     * @return the array list of visits
     */
    public static ArrayList<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 ArrayList<Transaction> getTransactions() {
        return transactions;
    }
}
```