**Spring Boot Bibliography Management System**

Keith Rodgers

5/15/2017

Software Design Patterns

# Problem Statement

Need to implement a bibliography managements system, similar to JabRef, as a web application. Bibliographies list the sources referenced in a research paper, article, etc. These sources may be referenced multiple times in articles so it is useful to have a system which stores such records and can be accessed quickly for use.

# Approach

To implement the system, I used the following technologies/frameworks:

## Application

*Spring Boot* – this framework provides programmers the ability to launch a web application in a short amount of time

*Java* – programming language to support requests to server

*MVC* – this application used model, view, controller pattern for implementation. The view was simple HTML while a controller/model, written in Java, supported requests from front end.

## Database

*MySQL* – open source relational database management system which allows data persistence for the application by storing all bibliography records

## Deployment

*Maven* – build automation tool used to deploy, configure, and import open source repositories. Java repositories could be easily added to pom.xml

# Implementation

## Structure

## Model

**BibE** – this class represents the model of the system, the bibliography entry. It consists of several fields, standard get/set methods, and some supporting methods to assist with representing the model in different formats

## View

The following html pages is the front end for the user. The parameters passed by the view, such as checkboxes or data entry, are passed to the controller for handling.

**Bibliography.html**

**Edit.html**

## Controller

**HomeController** – this class handles requests from the client and helps to bind data to the model. For each of the controller methods, implementation details were encapsulated in other helping methods so that the controller remained simple.

## Features

### Add / Edit / Delete

Each of the features were implemented in the DatabaseHelper class where each method represented a different type of SQL statement (e.g. select, insert, update). To provide more flexibility, some methods took ‘id’ as a parameter, either a list or single integer, which was passed from the View layer. Also, insert/update statements were genericized, with help from the model class’s methods, so that if a new field is added, these queries do not need to be updated.

### Import

This feature, implemented in BibEntryImporter class, takes a file as input, parses it using the jbibtex library, and maps to the model’s fields.

### Export

This feature, implemented in BibEntryExporter class, exports a selected entry to BibTeX format.

### Search

On the UI, the user is able to search for records through an input box. This feature, implemented in the View as a javascript function, searches the list of records in real-time.

### View IEEE formatted citation

An important use case this system supports is viewing the citation in a format which can be easily extracted and used in a bibliography.

I identified the citation formatting classes/methods implemented by JabRef and used them for my own use since formatting is extremely difficult. All supporting classes are in the CitationFormatting package. To use these classes, I needed to map my model to JabRef’s data model. For the purposes of this project, I limited the citation style to just IEEE, but it could be extended to other formats, such as MLA, simply by passing the desired format as a parameter from the UI and referencing the supported XML stylesheet in code.

### Fetch/Import Records from IEEE api

This application also allows the user to search entries in the IEEE database and import them into the system. IEEE provides an api where if provided with a set of parameters, such as author or title, a request returns an xml document of entries. Once the api is called, xml elements are de-serialized and mapped to the data model’s (BibE) fields through annotations (e.g. @XmlElement). After de-serialization, the list of entries are exposed to the UI where the user can select an entry and import it into the system directly.

I also experimented with the implementation of the Decorator pattern for creating the api’s request url. This can be found in the apiUrlCreator package. Similar to the example in class, there’s a base url and several decorators (api parameters) which decorate the base url.