BoomBook BookStore
Coding Standards

1 Introduction

Coding Standards Document is a set of guidelines which are essential for writing readable, maintainable, reusable, consistent and reliable code. This document reflects the uniform habits that programmers should obey during the development of the project. The main concerns of the document can be listed as:

- Naming standards
- File organization
- Comments
- Coding conventions
- Whitespaces

Each of them is explained Coding Standards Specifications part and should be revised before coding.

2 Description

Coding Standards Document is essential to enhance the development process. The intention of document is having a common coding style by defining a natural language between programmers to obtain consistency. Consistency plays a key role from this view. A non-consistent (mixed) coding standard often causes more problems than a bad coding standard. Having a consistent code standard profits in many areas which some of them listed below.

- Developers can understand new code fragments more easier way. Even though the same programmer revising his/her own code, after some time passed it would be painful to understand what the code is actually doing. So readability and maintainability are increased.
- The people that new to that programming language don’t need to create their own coding conventions but rather just follow coding standards.
- It decreases the amount of time to debug since the code would be more readable.
- People make fewer mistakes in consistent environments
- It ensures the quality of the code in some way.
- It helps the creation of test scripts.
- It ensures that the packages and files are well-resided and clean which is another essential feature that products should have.

- Less communication will be need between developers and other stakeholders because many answers of questions will be stated in coding standards document already.

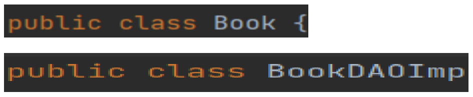
3 Coding Standards Specifications

3.1) Naming Standards

Naming Standards provides readability and maintainability by making the code more easy to read. Also it benefits the identification of a data type , variable and package.

3.1.1) Classes

Even thought class names can consist of single or multiple words , the compound word should be noun. In case multiple-word class name is used , the first letter of each unique word should be capitalized. Abbreviations are allowed as long as the descriptiveness of noun being remained.

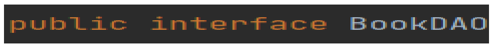


```
public class Book {
public class BookDAOImp
```

Figure-1 Class Samples

3.1.2) Interfaces

The names of interfaces should be noun. They can include multiple words and each single internal word's first letter should be capitalized. If the interface serves a widely used pattern such as data access objects then the common convention can be used in condition the noun should describe the task precisely.

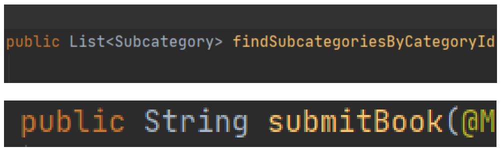


```
public interface BookDAO
```

Figure-2 Interface Sample

3.1.2) Methods

Even thought, there is no strict restriction about the type of words that will be used to create a method , it is necessary to describe the purpose of the method by name. It can include noun, verb or combination of them but should satisfy the description of method clearly. While using multiple-word names , the each unique internal word's first letter should be capitalized except the first one.



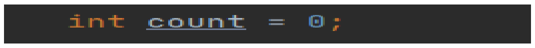
```
public List<Subcategory> findSubcategoriesByCategoryId
public String submitBook(@M
```

Figure-3 Method Samples

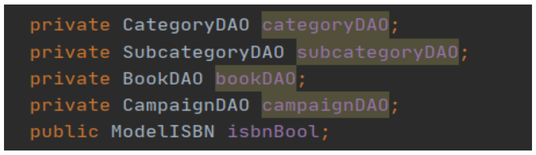
3.1.3) Variables

Variable names should describe the content that stored in them. It is preferred to keep that names short but it can depend on circumstances and the programmers are encouraged to make compromises on description over the length in that case. Variable names should start with lowercase whether they are single or compound. For concatenation of separate words the programmer has two options one of them is using a underscore (“_”) and the other one just concatenating words as they are but making the each word’s first letter capitalized except the first one.

Although it is not recommended, to increase the simplicity, the loop counter names can be letters but in condition the initialization of the counter should be internal of the loop declaration.



```
    int count = 0;
```



```
private CategoryDAO categoryDAO;
private SubcategoryDAO subcategoryDAO;
private BookDAO bookDAO;
private CampaignDAO campaignDAO;
public ModelISBN isbnBool;
```

Figure-4 Local and instance variable samples

3.1.4) Constants

As common convention the constants should be declared as fully uppercase words and there should be an underscore (“_”) between each unique internal word.

Note: In this project it is decided that there shouldn’t be any defined constant since it reduces the reusability of the components. The definition of the constants is stated in case there can be unordinary situations.

3.2) File Organization

3.2.1) File Naming Convention

Since the project uses MVC design pattern and implemented in Java language with the power of Spring Boot framework, the naming convention of files is critical. The widely-used naming convention for Spring Boot and MVC applications is decided while describing the functions of the classes. A comprehensive guide to naming convention is provided in below.

Java Source Files : Each java source file whether it is class or interface has a class in it and should be entitled with the same name as the class. So the class naming convention should be followed which is stated in naming standards.

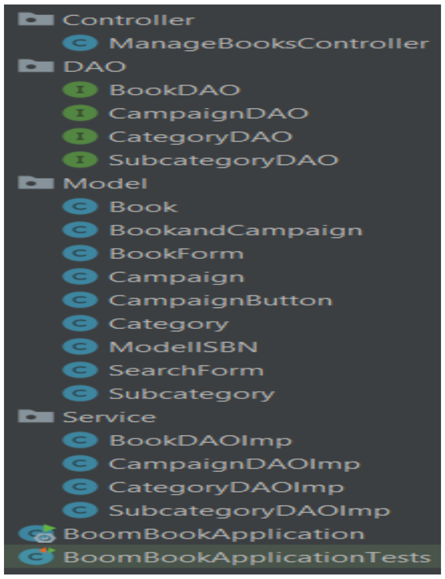

Controller
ManageBooksController
DAO
BookDAO
CampaignDAO
CategoryDAO
SubcategoryDAO
Model
Book
BookandCampaign
BookForm
Campaign
CampaignButton
Category
ModelISBN
SearchForm
Subcategory
Service
BookDAOImp
CampaignDAOImp
CategoryDAOImp
SubcategoryDAOImp
BoomBookApplication
BoomBookApplicationTests

Figure-5 Java Source File Naming Samples

Built-in Spring Configuration Files: Since the project application environment will be created automatically by Spring Boot Framework, there isn't any convention to entitle that sort of files such as xml files , properties files and Maven files. That file names should be untouched.

GUI Components: The GUI file names shouldn't contain special characters ($,~,& etc..) and can be any type of words like nouns , verbs etc. Also both uppercase and lowercase letters, multiple-words are allowed to use in any combination without one exception. The view files (HTML files) should start with lowercase letters and if consist of more than one word they should include dash ("-") between each o them. They should be simple and descriptive.

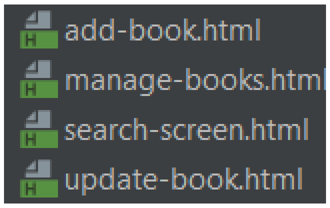

add-book.html
manage-books.html
search-screen.html
update-book.html

Figure-6 HTML Files Naming Sample

Packages: Package names should be nouns (single word names preferred) and they shouldn't include any special characters ($,~,& etc..). Although they can have any names, they can't have the same names as classes.

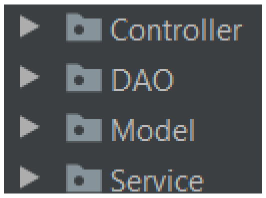

Controller
DAO
Model
Service

Figure-7 Package File Names

3.2.2) Organization

Beside the built-in structure provided by Spring Boot, inside the application folder there should be five essential components.

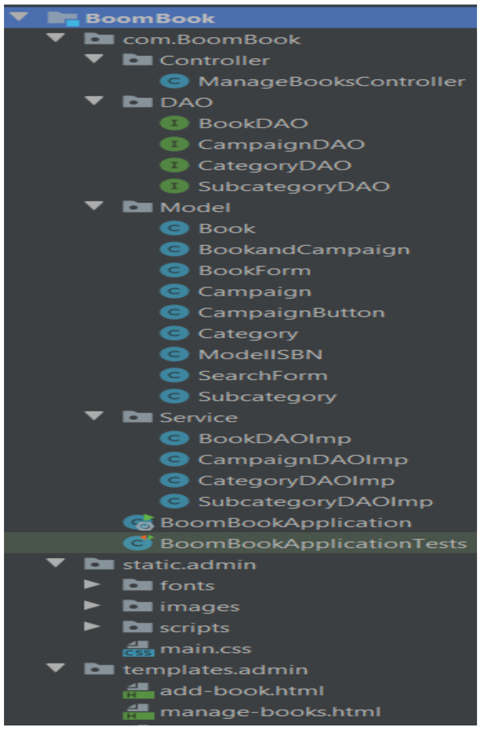
a) **Application Package:** This package should be entitled with project name with “com” domain. It includes Java source files with MVC structure. It contains mainly 4 packages and 2 Java files that are Controller package which the controller classes will be resided in , DAO package which will have data access object interfaces , Model package which consists of class files corresponding to tables in database, Service package that possess the implementations of interfaces in DAO package, BoomBookApplication class which runs the application from scratch and BoomBookApplicationTests class that has context loading test.

b) **Static Templates Package:** This package contains static view components such as fonts, externally downloaded UI frameworks, scripts and images.

c) **Dynamic Templates Package:** This package consists of view components of the application which are HTML files. The package can be divided into more parts in upcoming development periods. This package corresponds to view part of MVC architecture.

d) **Application Properties File:** This is a properties file which provides some important attributes such as database that will be connected , the port number that will be used on local server , username and password of the database etc..

e) **Libraries Package:** It is the package that consists of Spring built-in configuration files, embedded local server files, the files of dependencies added to the project, Maven files, plugin files and etc..



```
BoomBook
  com.BoomBook
    Controller
      ManageBooksController
    DAO
      BookDAO
      CampaignDAO
      CategoryDAO
      SubcategoryDAO
    Model
      Book
      BookandCampaign
      BookForm
      Campaign
      CampaignButton
      Category
      ModelISBN
      SearchForm
      Subcategory
    Service
      BookDAOImp
      CampaignDAOImp
      CategoryDAOImp
      SubcategoryDAOImp
    BoomBookApplication
    BoomBookApplicationTests
  static.admin
    fonts
    images
    scripts
    main.css
  templates.admin
    add-book.html
    manage-books.html
```

Figure-8 File Organization

3.3) Comments

There are three type of comments that can be written in project files. Two of them can be applied though Java files and the other one for view (HTML) files.

One of them is implementation comments in Java which are C++ type of comments delimited by “ /* …… */ “ and “ // ” . Other one is documentation comments that is specialized to Java that delimited by “ /**…… */ “. The last one for HTML type o files and delimited by “ <!-- --> “.

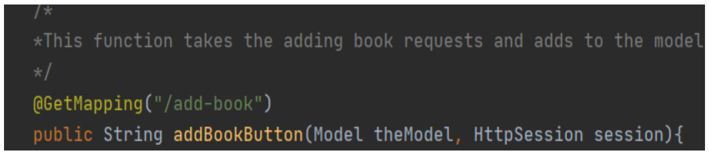
Comments should give only a overview of the code snippet rather giving a full detailed explanation. Programmer is responsible to write precise comments but in some cases it can indicate quality problems. So if the programmer feels obligation to write comments too many times in a code snippet, it is probably because of the poor quality of code.

Comments should never include special characters and complicated language. Also comments should be specific to that code fragment. On the other hand there isn’t any restriction about the syntax of the comments.

As known there are four types of comments as block, single-line, trailing and end-of-line.

3.3.1) Block Comments

Use this kind of comment to only give a specification or description for files, classes, methods, created complex data structures and algorithms. This comment type should be stated in the beginning of them. The general comment considerations mentioned above also works for this type of comment. In addition programmers shouldn't write these type of comment at the end of code fragments that need explanation. Only the beginning of the code is allowed to comment for this type.

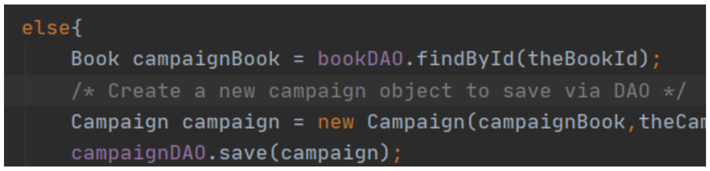


```
/*
*This function takes the adding book requests and adds to the model
*/
@GetMapping("/add-book")
public String addBookButton(Model theModel, HttpSession session){
```

Figure-9 Block Comments Sample

3.3.2) Single-line Comments

Use this kind of comments only inside the code fragment. Other usage of them is not allowed to prevent comment conflicts.

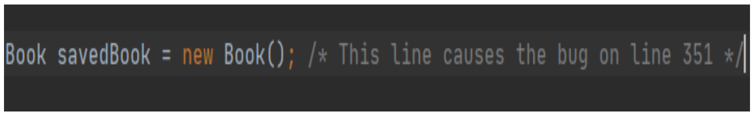


```
else{
    Book campaignBook = bookDAO.findById(theBookId);
    /* Create a new campaign object to save via DAO */
    Campaign campaign = new Campaign(campaignBook,theCa
    campaignDAO.save(campaign);
```

Figure-10 Single-Line Comments Sample

3.3.3) Trailing Comments

Distinctively in this project, programmers can use this kind of comments to keep track of their improvements. They can use them to take brief notes about where did they stay on last coding session or to inform other programmers about the code by indicating where the problem is , what needs to be improved etc..

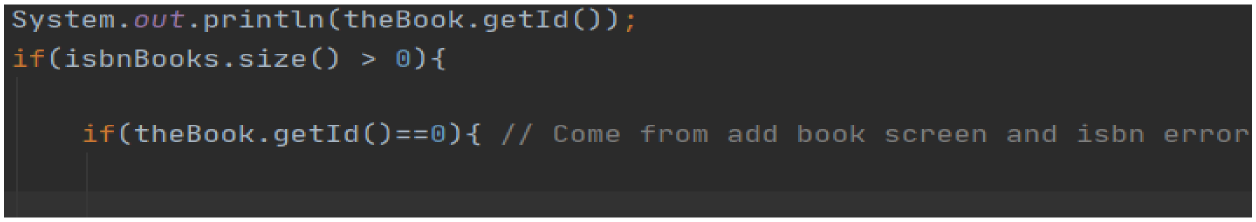


```
Book savedBook = new Book(); /* This line causes the bug on line 351 */
```

Figure-11 Trailing Comments Sample

3.3.4) End-Of-Line Comments

The programmers are totally free to use End-Of-Line comments with any type of words and intentions since it is the most practical way of making comments.



```
System.out.println(theBook.getId());
if(isbnBooks.size() > 0){

    if(theBook.getId()==0){ // Come from add book screen and isbn error
```

Figure-12 End-Of-Line Comments Sample

Note: Although programmers can write documentation comments, they are discouraged to use them since they are not necessary for this project and they can led to confusion.

3.4) Coding Conventions

Code-level conventions help programmer to eliminate poor quality code. It includes general assumptions and best practices on code-level optimization. Although they are stated in code standards and needed to be taken care of, they are not the only implementation rules on practice.

Here are some considerations:

- The fields of any instance or class can not be stated as public.
- The database interactions are made by data access objects. Don't try to access the database in another ways.
- Since the design pattern is determined as MVC, don't try to get HTTP requests without controllers.
- Don't use global variables and constants.
- Don't use static variables or methods.
- Don't assign multiple variables to the same value on the same line.
- Don't rely on operator precedence and use parenthesis to increase readability.
- Avoid empty catch blocks.
- Try to use finally blocks as often as possible
- Check method arguments before actually processing them to avoid NullPointerException.
- Keep source file length as small as possible.
- Use Java annotations even if some of them aren't mandatory
- Don't use any inner class
-

3.5) Whitespaces

3.5.1) Blank Lines

The conditions that always need at least one blank line is listed below:

- After every end of instruction (";") symbol,
- Between each HTML tags
- After class and interface declarations

- Between methods
- Between sections of source file
- After if-else statements

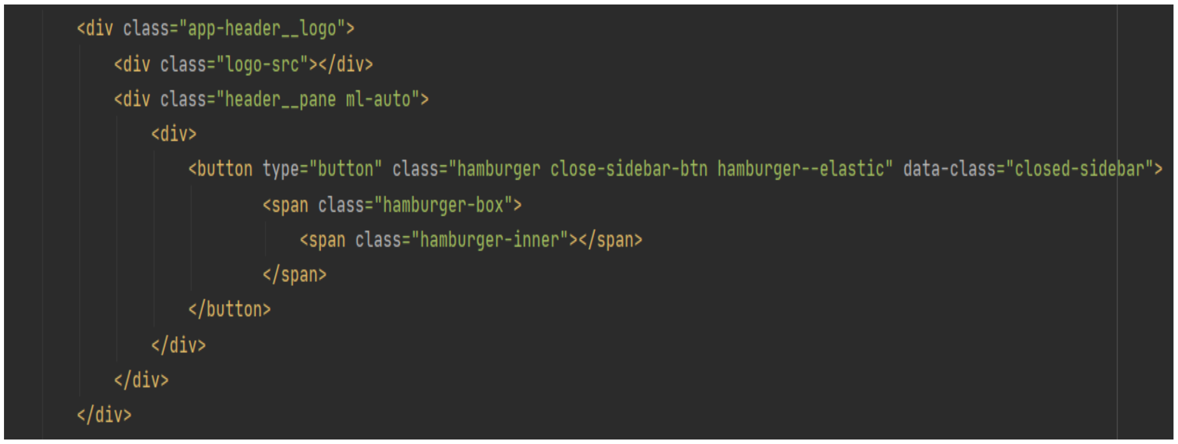
3.5.2) Blank Spaces

The conditions that always require at least one blank line is listed as:

- In nested HTML tags after blank line
- After loop , if-else statement, method declarations (just after the new blank line)
- After commas in function arguments
- Between the expressions in the for and while loops
- Just after doing the type casting operation

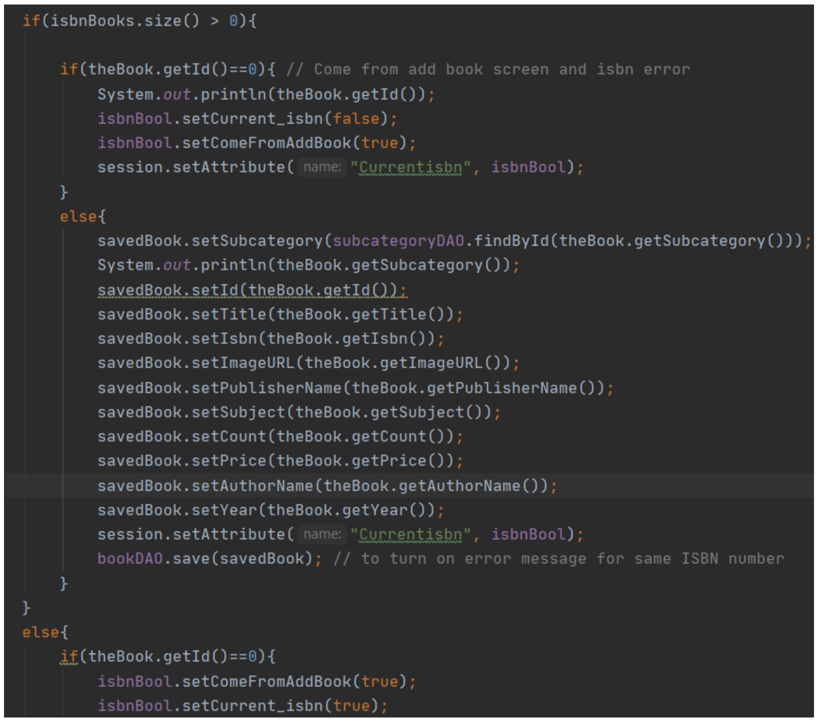
3.5.3) Indentation

Indentation means 4 consecutive whitespaces. In this project they are used to breaking long lines into smaller ones by making one right-hand side indentation each time. Also they are useful while breaking function calls and long conditions inside if statements.



```
<div class="app-header__logo">
    <div class="logo-src"></div>
    <div class="header__pane ml-auto">
        <div>
            <button type="button" class="hamburger close-sidebar-btn hamburger--elastic" data-class="closed-sidebar">
                    <span class="hamburger-box">
                        <span class="hamburger-inner"></span>
                    </span>
            </button>
        </div>
    </div>
</div>
```

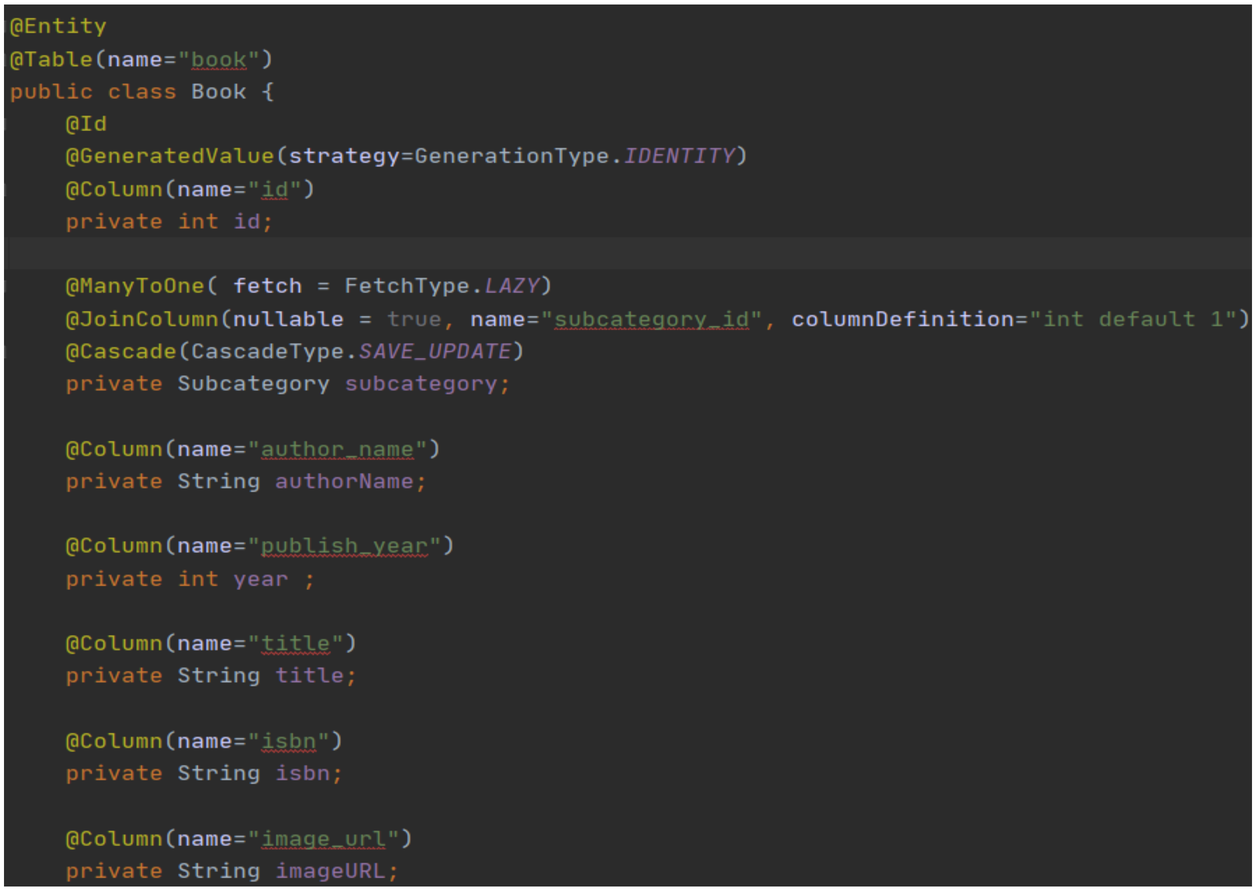
Figure-13 HTML Whitespaces Sample



```
if(isbnBooks.size() > 0){

    if(theBook.getId()==0){ // Come from add book screen and isbn error
        System.out.println(theBook.getId());
        isbnBool.setCurrent_isbn(false);
        isbnBool.setComeFromAddBook(true);
        session.setAttribute( name: "Currentisbn", isbnBool);
    }
    else{
        savedBook.setSubcategory(subcategoryDAO.findById(theBook.getSubcategory()));
        System.out.println(theBook.getSubcategory());
        savedBook.setId(theBook.getId());
        savedBook.setTitle(theBook.getTitle());
        savedBook.setIsbn(theBook.getIsbn());
        savedBook.setImageURL(theBook.getImageURL());
        savedBook.setPublisherName(theBook.getPublisherName());
        savedBook.setSubject(theBook.getSubject());
        savedBook.setCount(theBook.getCount());
        savedBook.setPrice(theBook.getPrice());
        savedBook.setAuthorName(theBook.getAuthorName());
        savedBook.setYear(theBook.getYear());
        session.setAttribute( name: "Currentisbn", isbnBool);
        bookDAO.save(savedBook); // to turn on error message for same ISBN number
    }
}
else{
    if(theBook.getId()==0){
        isbnBool.setComeFromAddBook(true);
        isbnBool.setCurrent_isbn(true);
```

Figure-14 Java Source Files Whitespaces Sample



```
@Entity
@Table(name="book")
public class Book {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @ManyToOne( fetch = FetchType.LAZY)
    @JoinColumn(nullable = true, name="subcategory_id", columnDefinition="int default 1")
    @Cascade(CascadeType.SAVE_UPDATE)
    private Subcategory subcategory;

    @Column(name="author_name")
    private String authorName;

    @Column(name="publish_year")
    private int year ;

    @Column(name="title")
    private String title;

    @Column(name="isbn")
    private String isbn;

    @Column(name="image_url")
    private String imageURL;
```

Figure-15 Java Source Files Whitespaces Sample