**Building Electronic Commerce System**

CSCI7795.81

**Online DK Bookstore**

Fall 2019

Instructor: Dr. Yongming Tang  
**by**

Dhara Patel [ID: 1820905]



**Fairleigh Dickinson University**

|  |  |
| --- | --- |
| **LIST OF CONTENTS** | |
| **SR NO.** | **TITLE** |
|  | **Online DK Bookstore** |
| **1** | **Introduction** |
| **2** | **Requirements of the e-Commerce Project** |
| **3** | **Data Modeling** |
| **4** | **System Design** |
| **5** | **Implementation Strategy** |

|  |
| --- |
| **Online DK Bookstore System** |

|  |
| --- |
| **1. Introduction** |

Online DK Book Store is a website that sells books online using Java Servlet, JSP and Hibernate framework. DK Book Store has a web application where the customer can purchase books online. For creating this website I choose Eclipse, which is an integrated development environment(IDE) in this I used web dynamic project with the most widely used java IDE. I used Java Servlet, Hibernate Framework with JPA, MySQL database, J UNIT testing on back-end side and JSP, JSTL, HTML, CSS, JavaScript and j Query, Bootstrap 3, Lucid-chart on front-end side. I used tomcat server to run the website over the browser and also using MySQL database for storing data in database. I used Lucid-chart for drawing the UML diagrams.

When customers enter the website URL on browser then home page will be loaded, where customer can see all listings of books in DK Book store. Customers can search for a book by its title, author and description, customer can view the book details from the view book detail page. Customer can add book to the shopping cart and finally purchase using credit card payment transaction. An e- mail notification is sent to the customer email address. The customer can login using his account details or new a customer can register very quickly. They should give the details of their name, contact number and shipping address. The customer can also give feedback to a book by giving rating on a rating system from zero to five. The books are divided into many categories based on subjects like Java Programming, Software Engineering, or Database Systems etc.

On employee side, Employees are able to login with their employee ID via sign up functionality. They are able to add books, update/delete books and manage order status. They also can manage order page where they can search the order by id, order date or order status.

Before coding I used J UNIT test framework to write and run tests for testing data. Unit testing is an important part in Test Driven Development (TDD) as it helps finding problems in the code as early as possible, especially when you make changes to the existing code, you can run unit tests again to make sure that the changes do not break the application.

It has following functionalities: **Sign Up** (customers and employees can register in the DK Bookstore system, **Login** (customers or employees can go to pages with respective functionalities on the DK Bookstore system website), **Manage Payment** (Customers can add multiple payments to their account), **Search Books**(customers can search for specific books by title, author, description), **View Books** (customers can view detail about a specific book), **Add To Cart**(customers can add one or multiple items to the cart), **Update Cart** (customers can modifies the sub quantity of a book or can remove item from shopping cart), **Write Review** (customers can create reviews for the each book), **Checkout**  (customers can checkout and place the orders), **Add Book**(Employees can add the books),**Update book** (Employees can update the book information),**Delete Book**(Employee can delete book),**Clear Cart**(Customer can clear cart from shopping cart if they don't want to buy any book), **View Order**(Customer can view order details which customer already ordered books from DK bookstore system),**Manage Order** (Employees can manage orders by order id or Dates or status),**Logout** (customers and employees can log out of the system).

|  |
| --- |
| **2. Requirements of the e-Commerce Project** |

**2.1 Application requirements: Describe all the functionalities for employees and customers.**

**Functionalities for Customers:**

1. SignUp (Customer)

2. Login (Customer)

3. ManagePayment (Customer)

4. SearchBooks (Book)

5. ViewBook (Book)

6. AddToCart (Shopping Cart)

7. UpdateCart (Shopping Cart)

8. ViewCart (Shopping Cart)

9.ClearCart(Shopping Cart)

10. Checkout (Shopping Cart, Order)

11.viewOrder(Shopping Cart, Order)

12. WriteReview (Review)

13. Logout (Customer)

**Functionalities for Employees:**

1. SignUp (Employee)

2. Login (Employee)

3. AddBook (Book)

4. UpdateBook (Book)

5. DeleteBook(Book)

6. ManageOrderStatus (Order)

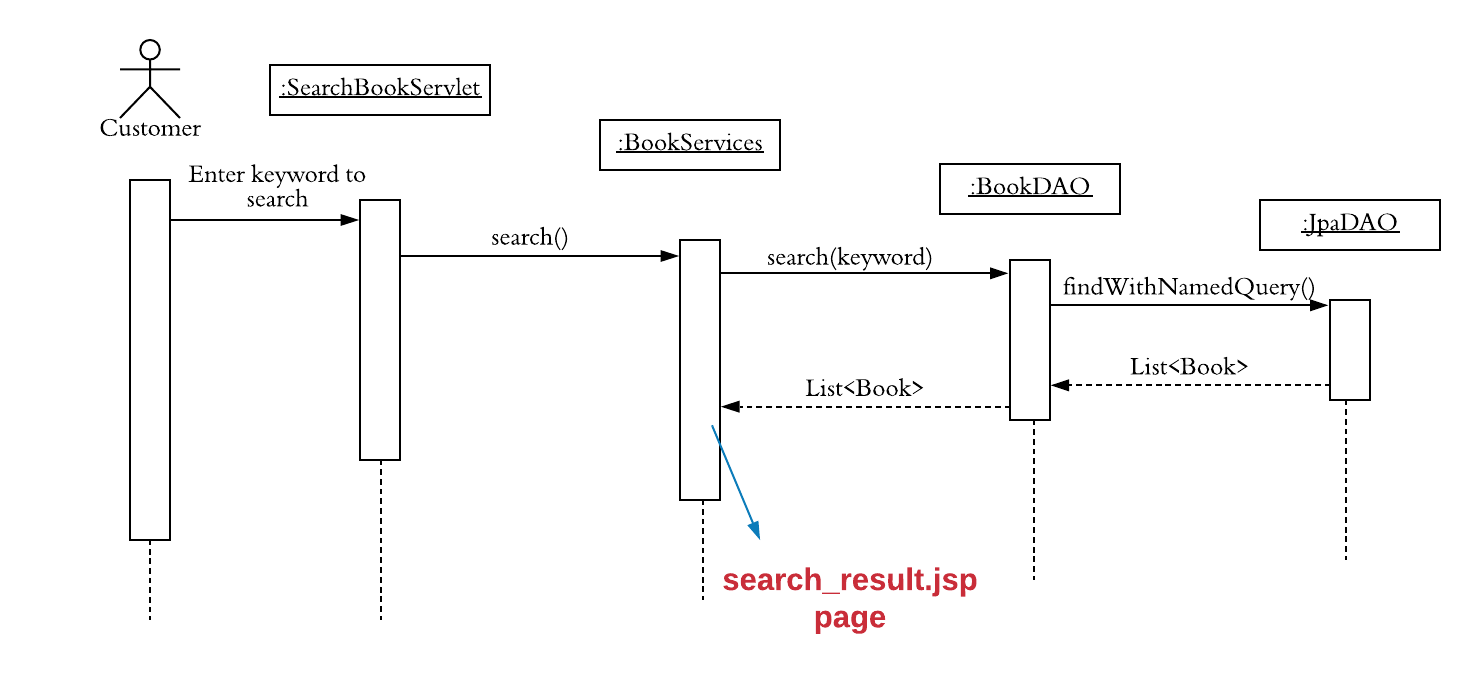
7. Logout (Employee)

**2.2 Refine Use Case of all Functionalities**

**2.2.1 A customer is able to do search books.**

**Dynamic Model:**

**Sequence Diagram of Search Book**

**[Fig 1: Sequence diagram of Search Book]**

Functionality: Customers search the books.

Functionality Name: **Search Books**

**Functional Model:**

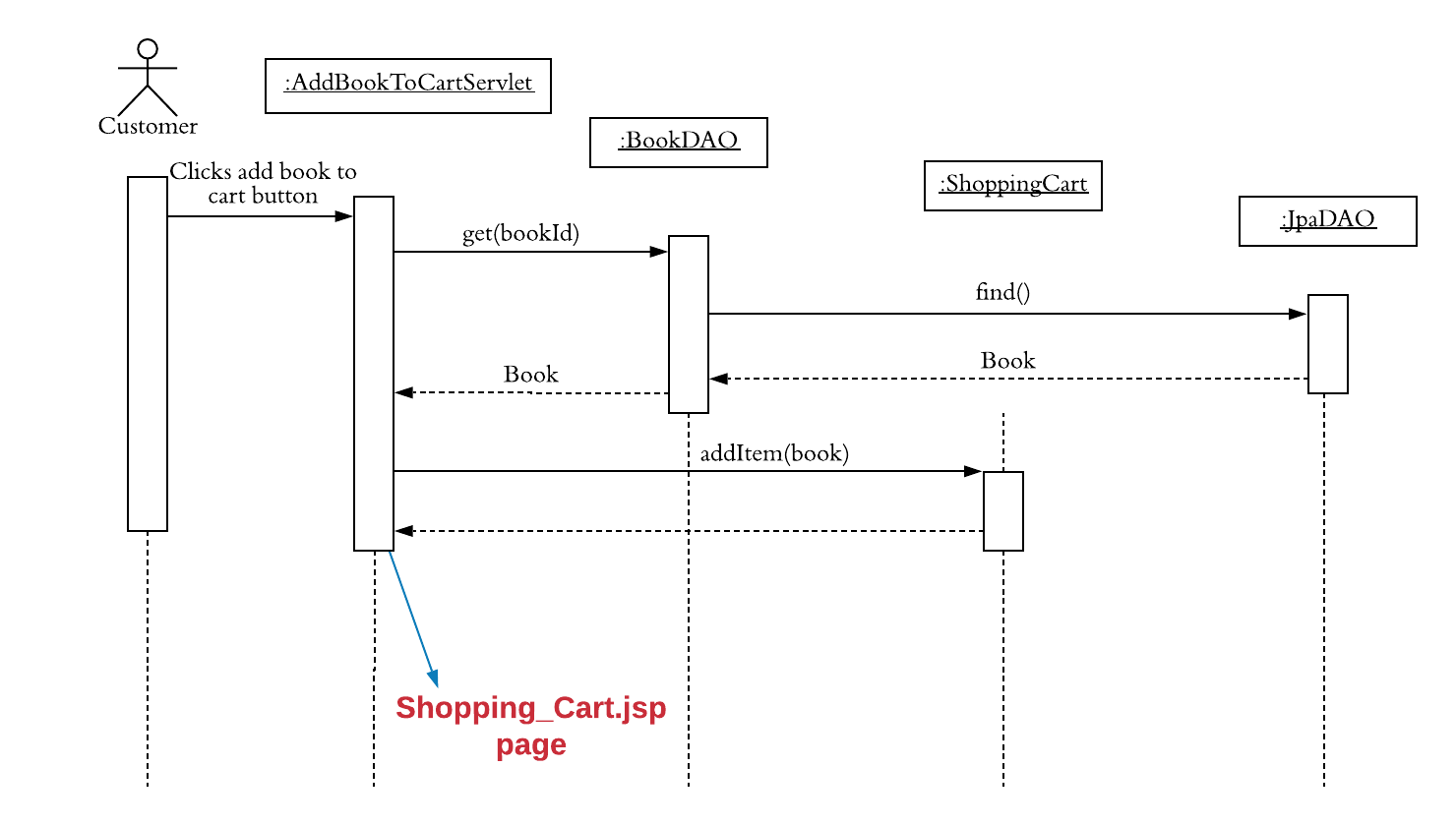
**Refine Use Cases**

|  |  |
| --- | --- |
| ***Use case Name*** | SearchBooks |
| ***Participating Actor*** | Initiated by a Customer |
| ***Flow of Events*** | 1.The customer enter keyword to search with specific title, name, description, author in search bar.  2.SearchBookServlet invoked and it calls the search() method on the Bookservices class then calls the search() method on the BookDAO class then, invokes the findWithNamedQuery() method of the JpaDAO class.  3.The result returned is a List collection of Book objects.  4.The BookServices class forward the request to the search\_result jsp page. |
| ***Entry conditions*** | The customer has loaded the home page. |
| ***Exit conditions*** | A confirmation is shown. |
| ***Quality Requirements*** | The process must be done in less than 3 seconds. |

**2.2.2 A customer is able to do add the books to the shopping cart.**

**Dynamic Model:**

**Sequence Diagram of Add To Cart**

**[Fig 2: Sequence diagram of Add To Cart]**

Functionality: Customers add the books to the shopping cart.

Functionality Name: **Add To Cart**

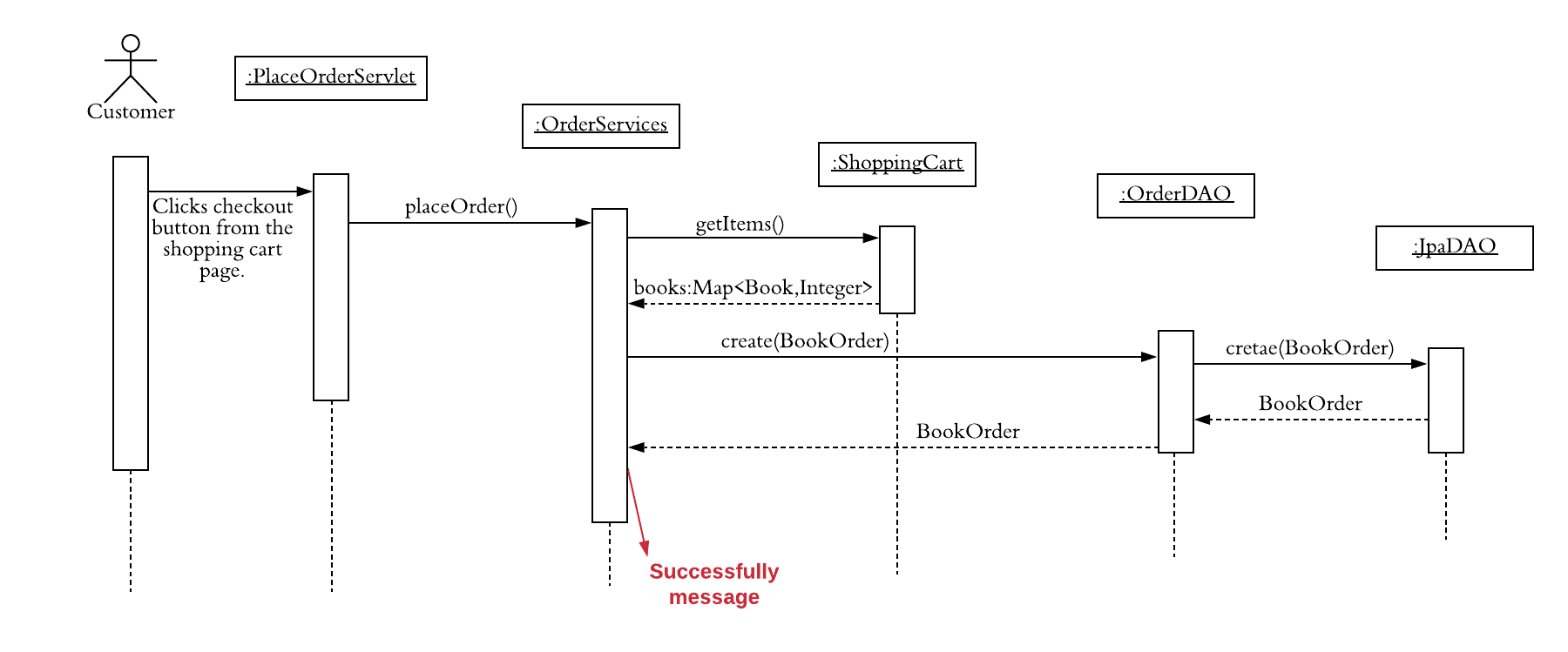
**Functional Model:**

**Refine Use Cases**

|  |  |
| --- | --- |
| ***Use case Name*** | AddToCart |
| ***Participating Actor*** | Initiated by a Customer |
| ***Flow of Events*** | 1.The customer viewed the specific book details, the customer adding the book into shopping cart by clicking on the Add To Cart button on home page or category drop down list.  2.The customer clicks Add To cart button and instance of the AddToBookservlet is created and it invokes the get(bookId) method on the BookDAO class to retrieve book object from the database and then it caused addItem(book) method on the ShoppingCart class to put the ShoppingCart and then forward the customer to the shopping\_cart jsp page. |
| ***Entry conditions*** | The customer has viewed the books item. |
| ***Exit conditions*** | The book is added to the shopping cart. |
| ***Quality Requirements*** | The process must be done in less than 3 seconds. |

**2.2.3 A customer is able to do checkout.**

**Dynamic Model:**

**Sequence Diagram of Checkout**

**[Fig 3: Sequence diagram of Checkout]**

Functionality: Customers checkout from the shopping cart.

Functionality Name: **Check Out**

**Functional Model:**

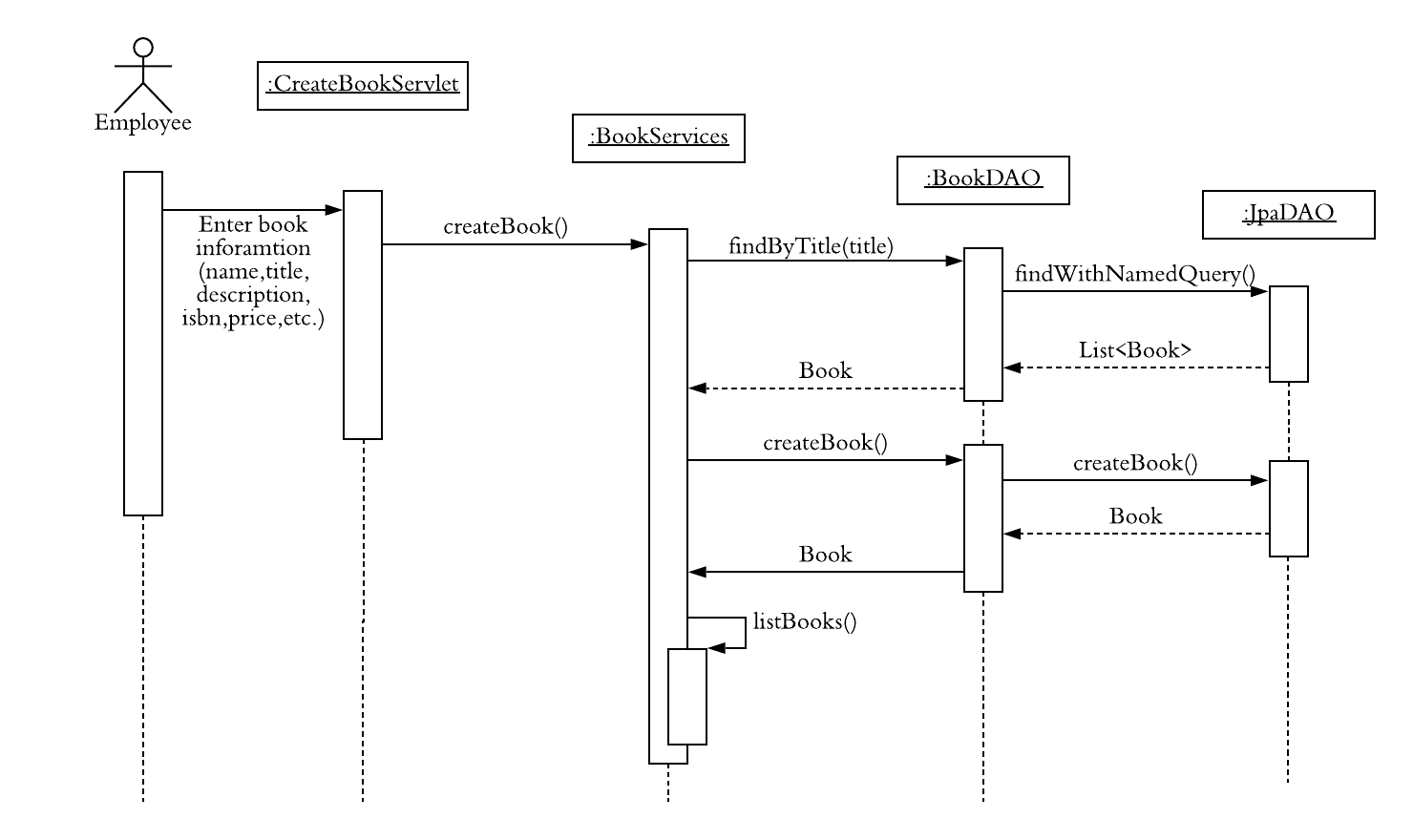
**Refine Use Cases**

|  |  |
| --- | --- |
| ***Use case Name*** | CheckOut |
| ***Participating Actor*** | Initiated by a Customer |
| ***Flow of Events*** | 1.The customer is processing to checkout by clicking on proceed to checkout button in shopping cart page.  2.The customer view the checkout page and instance of PlaceOrderservlet class created.  3.This class calls the placeOrder() method on instance of OrderService class and it getItems() from the Shoppingcart object to read all books content in the ShoppingCart and then invokes a create(BookOrder) methos on the OrderDAO class to accept the order to the database.  4.Finally, Display a successfully message with Order summary in orderSummary page. |
| ***Entry conditions*** | The customer has added the books to the shopping cart. |
| ***Exit conditions*** | The order placed successfully message shown with order summery or unsuccessfully message shown. |
| ***Quality Requirements*** | The process must be done in less than 3 seconds. |

**2.2.4 A employee is able to do add the books.**

**Dynamic Model:**

**Sequence Diagram of Add Book**

**[Fig 4: Sequence diagram of Add Book]**

Functionality: Employees add the books.

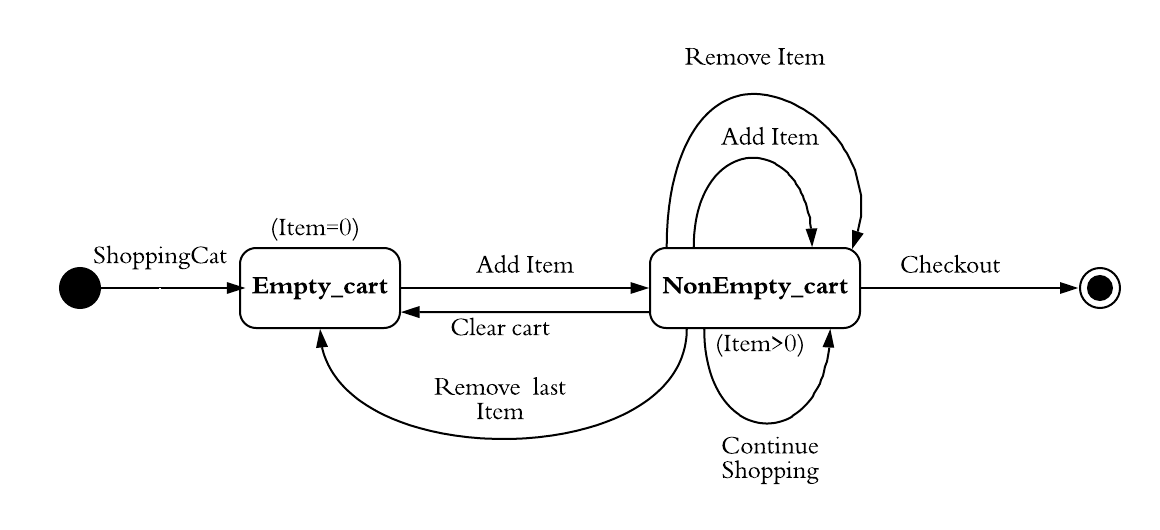
Functionality Name: **Add Book**

**Functional Model:**

**Refine Use Cases**

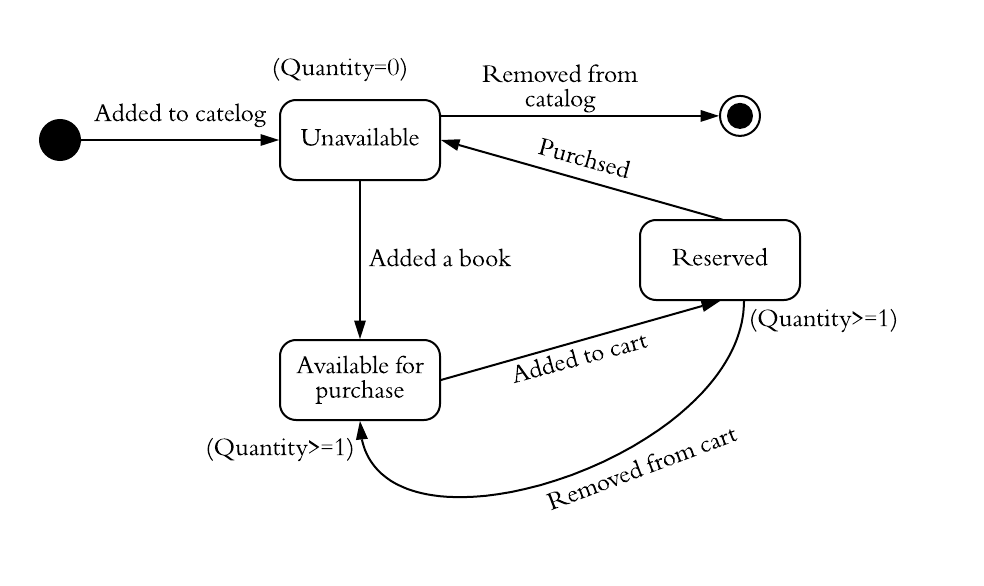
|  |  |
| --- | --- |
| ***Use case Name*** | AddBook |
| ***Participating Actor*** | Initiated by an Employee |
| ***Flow of Events*** | 1.The employee logged into his/her system and adding book information on book form page.  2.The createBookServlet class calls the createBook() method on the BookServices class and BookServices class invokes the findByTitle(title) method of the BookDAO class to find a book by title, invokes the findWithNamedQuery() method of the JpaDAO class if there’s a book exists with the same title and if not then we call the create(Book) method on the BookDAO class, then invokes the create(Book) method to the JpaDAO class. And returns the newly created Book object.  3.Finally, refresh the list book page with newly created book. |
| ***Entry conditions*** | The Employee is logged In into his system. |
| ***Exit conditions*** | The book is added in system. |
| ***Quality Requirements*** | The process must be done in less than 3 seconds. |

**State Diagram of Shopping cart object**

****

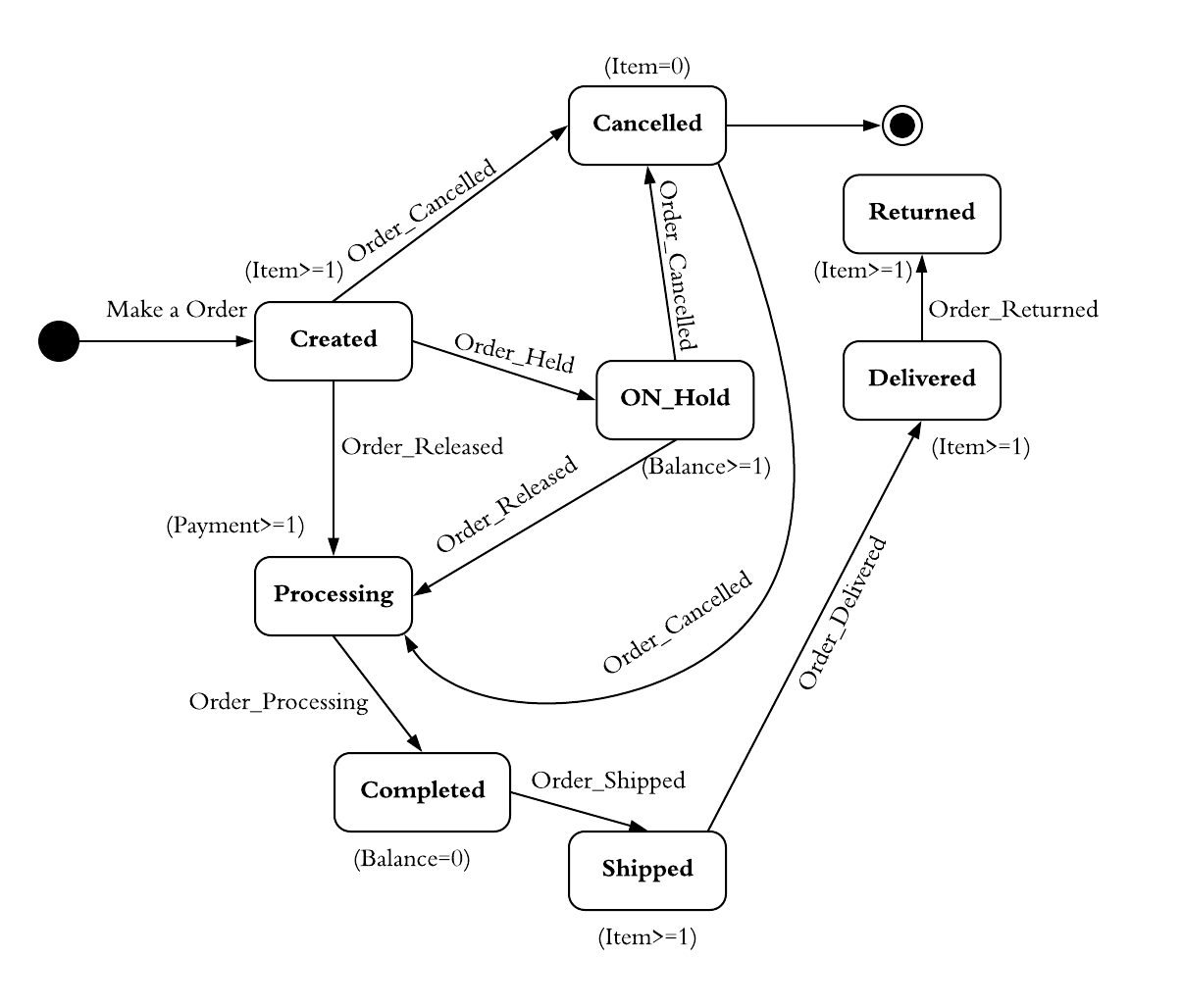
**[Fig 5:Dynamic Model:-Sate diagram of Shopping Cart Object]**

**State Diagram of Book object**

****

**[Fig 6:Dynamic Model:-Sate diagram of Book Object]**

**State Diagram of Order object**

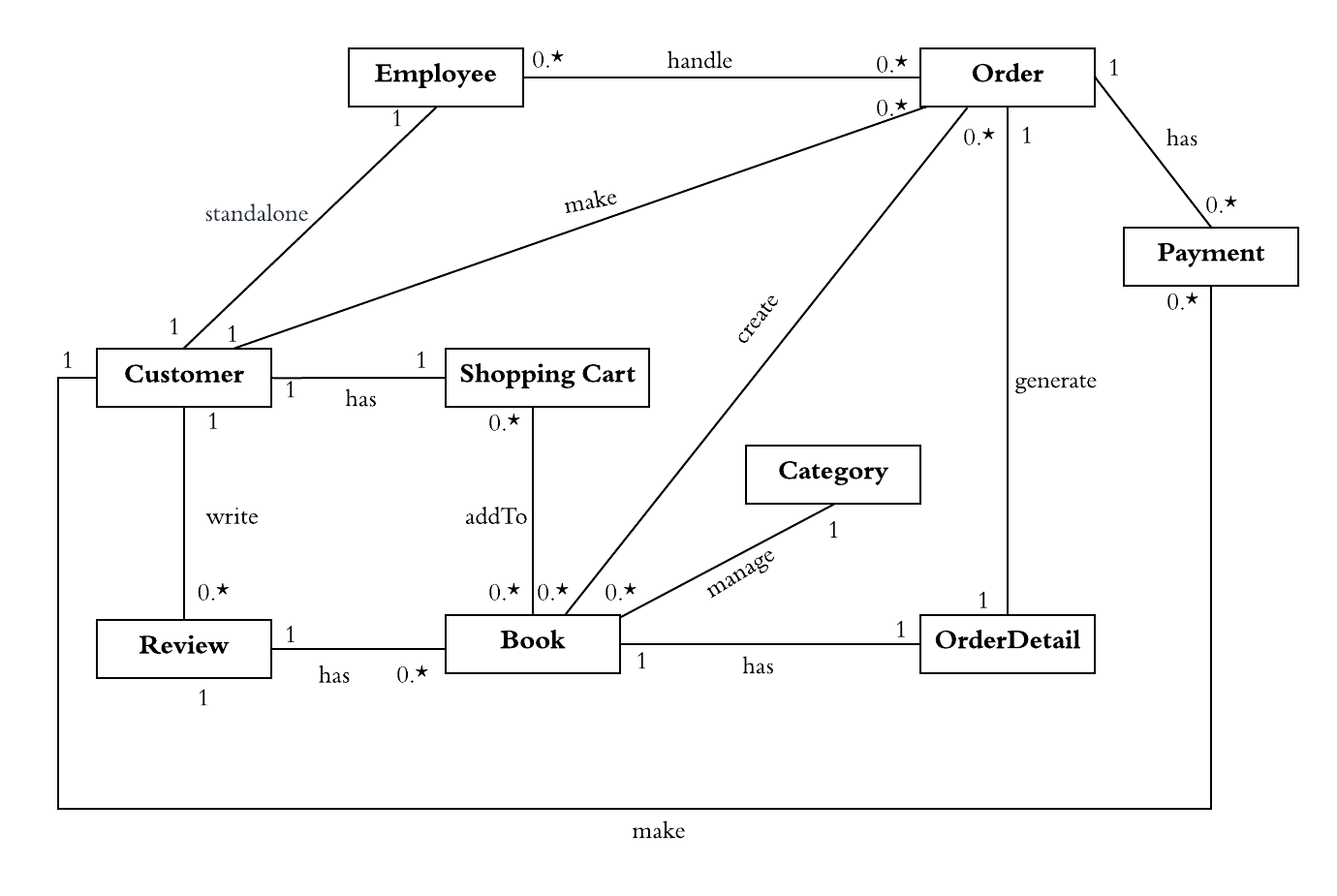
****

**[Fig 7:Dynamic Model:-Sate diagram of Order Object]**

|  |
| --- |
| **3. Data Modeling** |

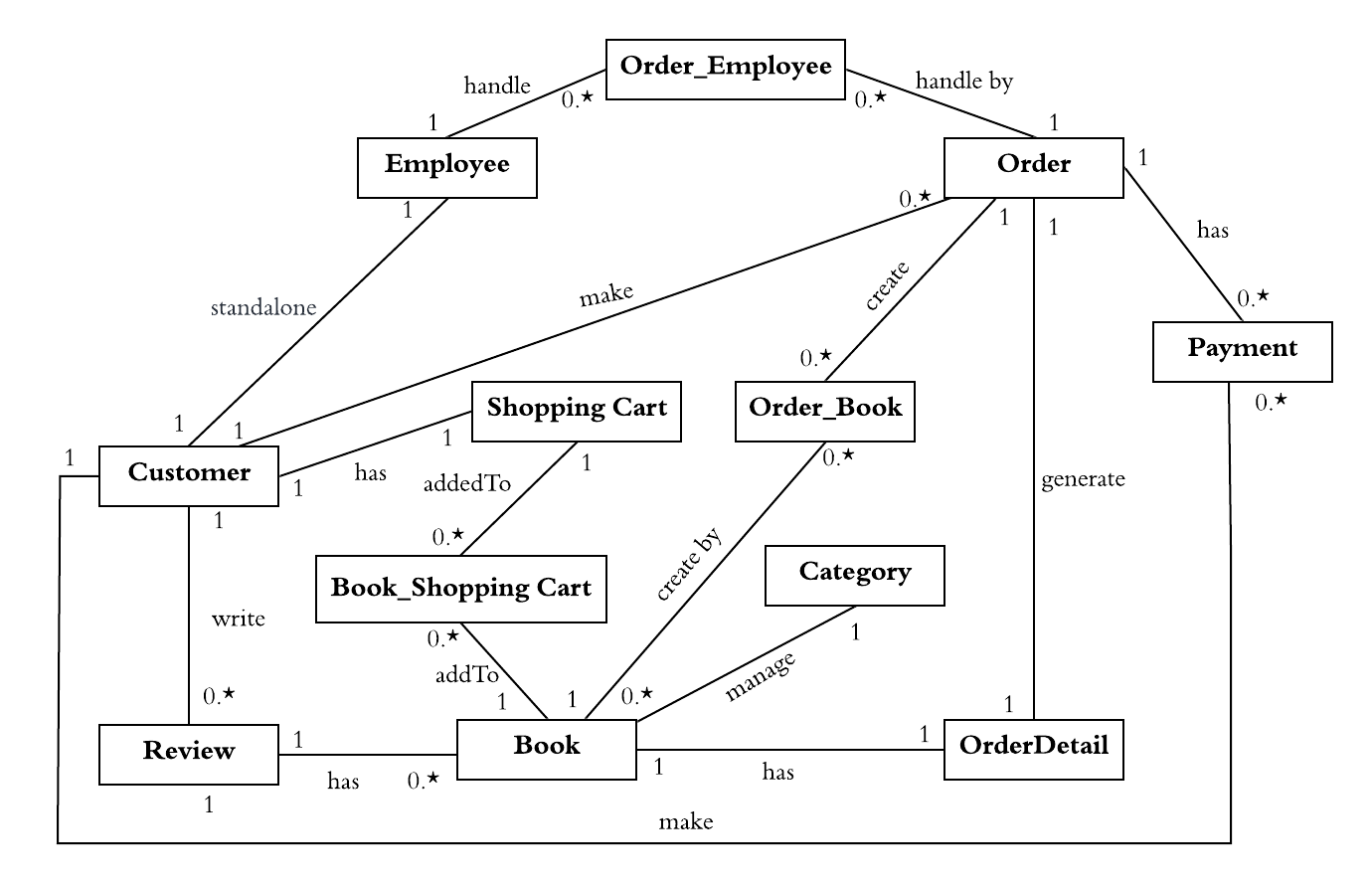
**3.1 Data entities and their relationships**

**3.1.1 Initial Class Diagram**

****

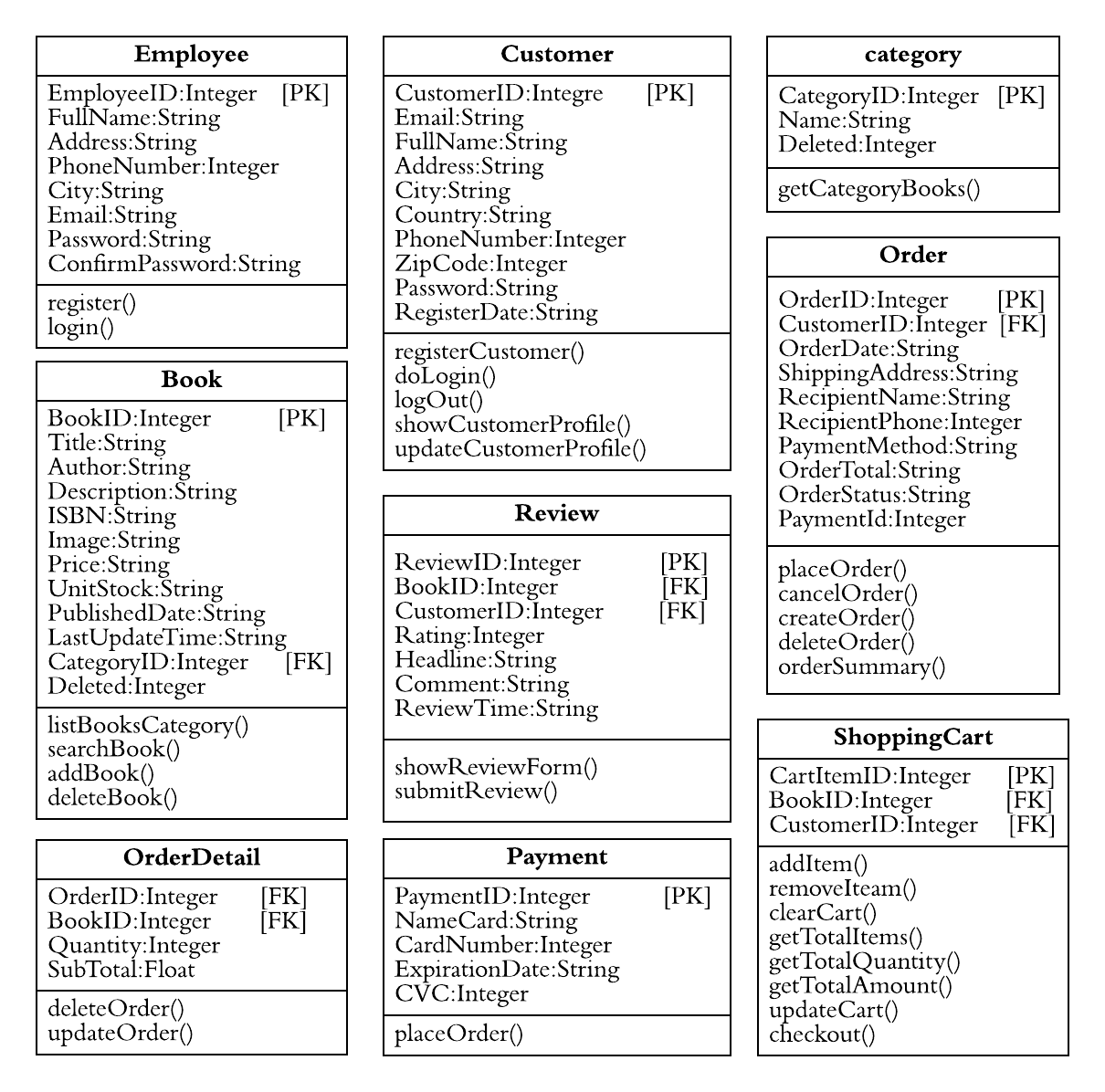
**[Fig 8: Initial Class Diagram]**

**3.1.2 Final Class Diagram**

****

**[Fig 9: Final Class Diagram]**

**Class, Attributes, Operation**

**[Fig 10: Class, Attribute, Operation Diagram]**

**3.2Final table schema**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table Name** | **Table Field** | **PK** | **FK** |
| Customer | CustomerID  Email  FullName  Address  City  Country  PhoneNumber  ZipCode  Password  RegisterDate | CustomerID |  |
| Category | CategoryID  Name  Deleted | CategoryID |  |
| Book | BookId  Title  Author  Description  ISBN  Image  Price  UnitStock  PublishedDate  LastUpdateTime  CategoryID  Deleted | BookID | CategoryID |
| Review | ReviewID  BookID  CustomerID  Rating  Headline  Comment  ReviewTime | ReviewID | BookID  CustomerID |
| *Order Table require two tables: master and detail.* | | | |
| Order(master) | OrderID  CustomerID  OrderDate  ShippingAddress  RecipientName  RecipientPhone  PaymentMethod  OrderTotal  OrderStatus  PaymentID | OrderID | CustomerID  PaymentID |
| Order Detail(detail) | OrderID  BookID  Quntity  SubTotal |  | BookID  OrderID |
| Payment | PaymentID  Namecard  CardNumber  ExpirationDate  CVC | PaymentID |  |
| ShoppingCart | CustomerID  CartItemID  BookID  GradTotal | CartItemID | CustomerID  BookID |
| Employee | EmployeeID  FullName  Address  PhoneNumber  City  Email  Password  ConfirmPassword | EmployeeID |  |

|  |
| --- |
| **4. System Design** |

**Front End:**

When customer enter the website URL**(**[**http://localhost:8080/OnlineBookStoreWebsite**](http://localhost:8080/OnlineBookStoreWebsite/)**/**[**)**,](http://localhost:8080/OnlineBookStoreWebsite/),the) the home page opens. The website lists all categories in the top drop down menu, which allows the customer to browse books in a specific domain and also search books by providing a specific keyword e.g. ‘effective java’,customer can view the all information about the each book and read reviews of other customers as well as average rating of a book, which can be ranged from 0 to 5 stars. customer can make only one review for a particular book but login is required first.

Before logging in, the customer must register an account by providing their personal information like full name, e-mail address, password, phone number, and the information required for shipping such as address, city, zip code and country. Customer can login by providing email and password.

Where customer are able to see the most recently published books (based on the publish date, not by the date on which the book is put onto the website). Customers can see the best-selling books (based on the number of orders have been made through the website), and the most favored books (based on their rating and number of reviews).

When customers are reading the details of book, the customer can add the book to their shopping cart by clicking the button **“Add to cart”**.Then shopping cart lists all the books that have been added. Customers can add a book multiple times to the shopping cart to increase the quantity(number of copies).The shopping cart page allows the customer to Update the quantity, remove books and even clear the cart, they also allow to keep continuous shopping. Customer can add shipping option(Free,3 Days, Next Days) delivery with sales tax(3%) of subtotal and shipping charge.

The information in shopping cart is maintained during the customer’s session, which means that she can continue navigating the site before placing an order. The website also provides a menu that allows the customer to see her shopping cart quickly and for convenience, customer doesn’t have to login first to use the shopping cart.

To place an order after adding books into shopping carts, the customer needs to review all books in the cart (customer can modify quantity or remove some books) before clicking “**Check Out”**. The checkout process requires the customer to login. If not, they are redirected to the login page, and the site proceeds to the check out page upon successful login.

On the Check Out page, the customer can review the books they want to order again, and confirm the shipping information. By default, the shipping information is filled with customer’s registered information (name, phone, address, city, zip code and country) and add the payment information credit card through. The customer can update this information if needed. The website accepts multiple payment Delivery.

The customer clicks **“Place Order”** button to submit the order to the employee staff, and then they can check the order status via **“Orders”** menu. And same time Order summary will be send to the customer logged in email address. The default status of an order is ‘Processing’ and only the employee’s staff can update the order status(canceled, shipped, Delivered, Completed, Returned).

On the Orders page, the customer can see all orders they have made through the website. The most recent orders are shown first and can click to see the details of each order. At this time, the customer can’t edit their orders once they are submitted.

The website also allows the customer to view their registered information (profile details) and edit it. Customer cannot change their registered e-mail address, and their password won’t be changed if left blank in the edit form.

when the customer logs out, the information in their shopping cart is cleared.

At any time, the customer can search the books they want by typing a keyword in the search box at the top of the site. The search result shows books that have either title or description contain the specified keyword. The customer can add the book to cart directly in the search result.

After inactivity the customer web page will expire in five minutes. And cookies will be store longer in session. Shopping cart information like subtotal, tax and shipping, Order total will be store in session

**Back End:**

When employee enters the URL**(**[**http://localhost:8080/OnlineBookStoreWebsite/admin/**](http://localhost:8080/OnlineBookStoreWebsite/admin/)**)**,

the home page is loaded, The employee can see recent customer ordered sales, recent customer reviewed books, how many books, categories, customers, reviews, total orders are in statistics panel on website home page.

Employee to manage information about categories, books, customers, reviews and orders with operations like view items (listing and detail); create, update, and delete an item. But they cannot create new review or new order.

The customer can manage all categories that are used to classify the books on the website. The only information needed for a category is the category name. Customer cannot delete a category if it contains books.

For adding a book onto the website, the employee can upload an image file that is used as the book’s thumbnail. When editing a book, employee can choose to update the thumbnail or not. And if not, the old image is kept. Employee cannot remove a book if there are reviews and orders on it.

For managing customers, customer’s email is unique. Employee can view the customer information.

For managing reviews, the employee cannot create a new one. Only customer can write a review. Employee can delete any review if it is inappropriate or violates DK bookstore website terms and conditions.

For managing orders, the employee can update shipping information and status of an order (more options for credit payment method).

After inactivity the employee web page will expire in five minutes. And Cookies will be stored in session.

|  |
| --- |
| **5. Implementation Strategy** |

* **Techniques used for your assignments from the front end to the back end including DB**

**Backend: -** Java Servlet, Hibernate Framework with JPA, MySQL database, JUNIT testing

**Frontend: -** JSP, JSTL, HTML, CSS, JavaScript and j Query, Bootstrap 3, Lucid-chart

**Database:** MySQL Server

* **Development Tools**

**IDE:** Eclipse 2019-06 [Dynamic Web Project]

**Java Application Server:** Tomcat 9.0

**Database:** MySQL Server

* **Hardware and software environments**

**Software environments:**

* I choose eclipse software to create dynamic web Project, Which is an integrated development (IDE) and is the most widely used Java SE(java 8) .
* Apache Tomcat version 9.0 to run Dynamic Web project.

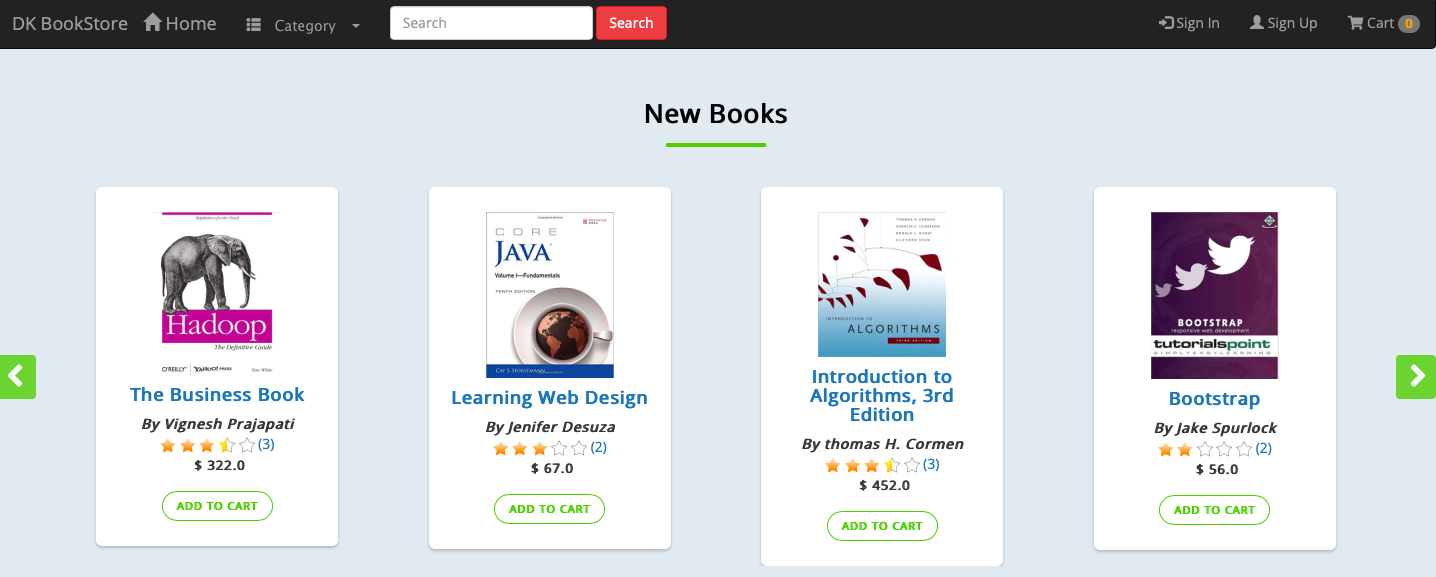
**Hardware environments:**

* MacBook Air

**Appendices if any**

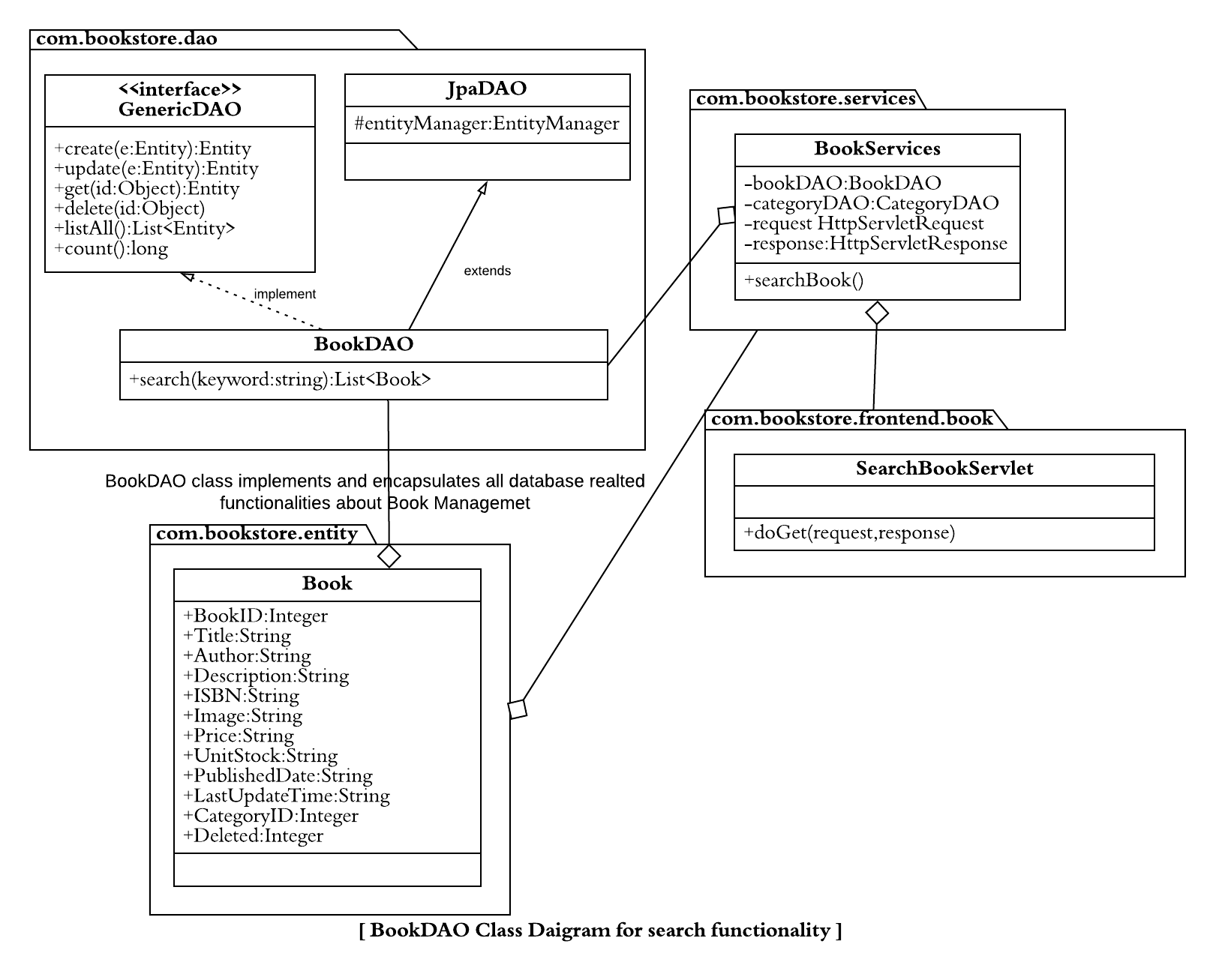
Below is the search book functionality from front end and back end perspective,

**Screenshot of customer search home page.**



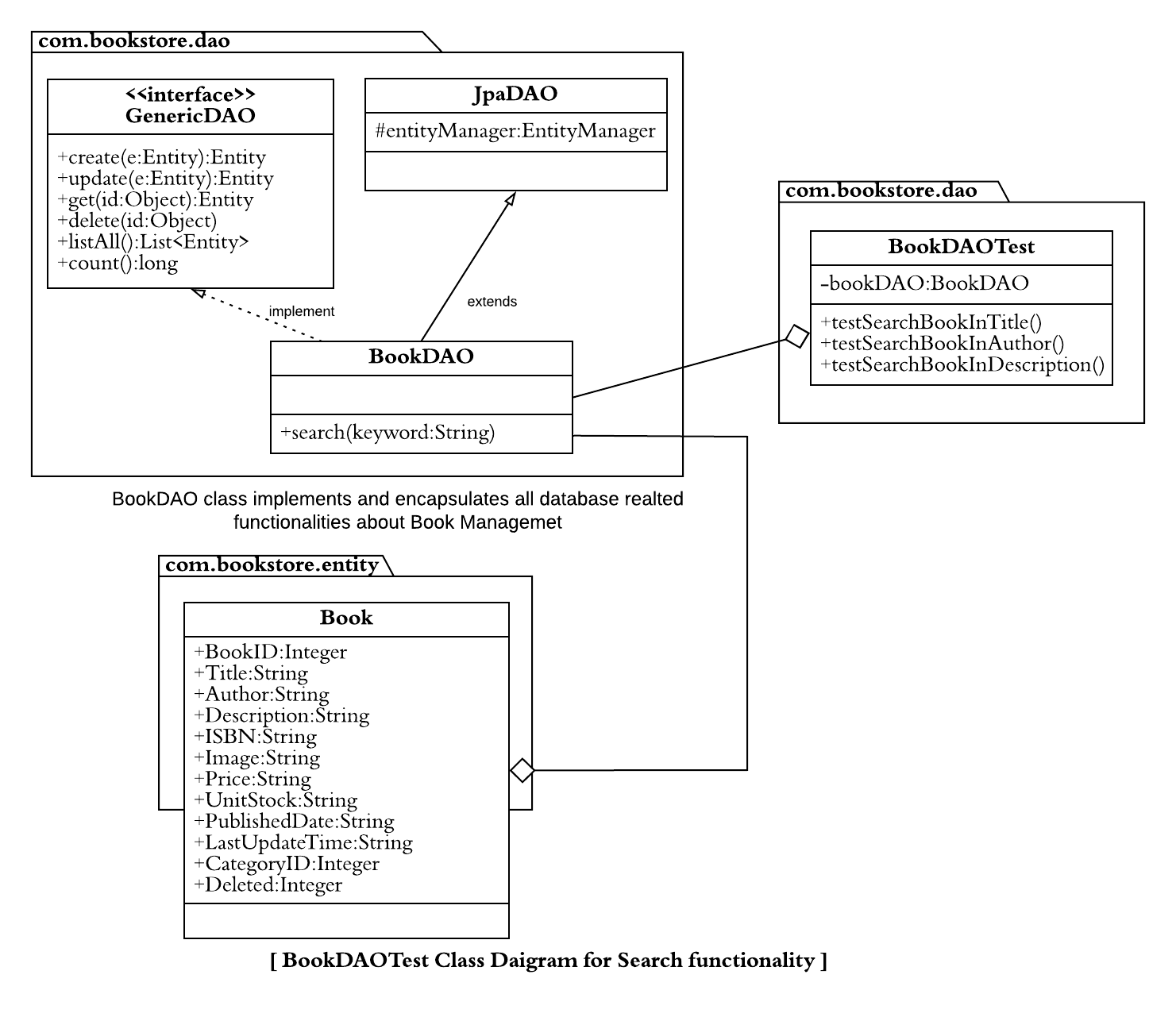
**BookDAO Class Diagram for Search Book Functionality**

In BookDAO class, Implement a search() method that method searches for books based on the keyword and returns a List collection of Book objects. In BookServices class, update to add the searchBook() method. In Controller layer, create the SearchBookServlet that handles search requests from the client.



**BookDAOTest Class Diagram For search functionality Testing**

First we creating BookDAO class and implement. Beside implementing the methods defined by the GenericDAO interface (create, update, delete, get, listAll, count).We also implement findByTitle, listByCategory, **search**, listrecentlyAdded methods in BookDAO class. Here, we implement search() method in BookDAO class. Book is domain model class which I mention in com.bookstore.entity. This class was generated by Hibernate Reverse Engineering tool. BookDAOTest class is a testing class for testing data before coding implementation which I mention in com.bookstore.dao. This class implement **testSearchBookInTitle(), testSearchInAuthor(), testSearchInDescription()**.



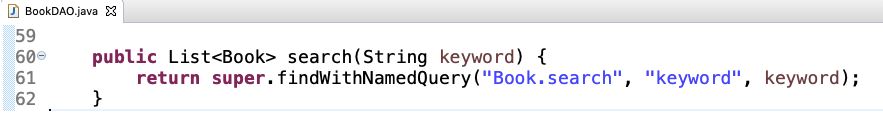
**J UNIT Testing for Search Functionality and implemntation**

**1. First Implement search() method in BookDAO .java and write unit tests**

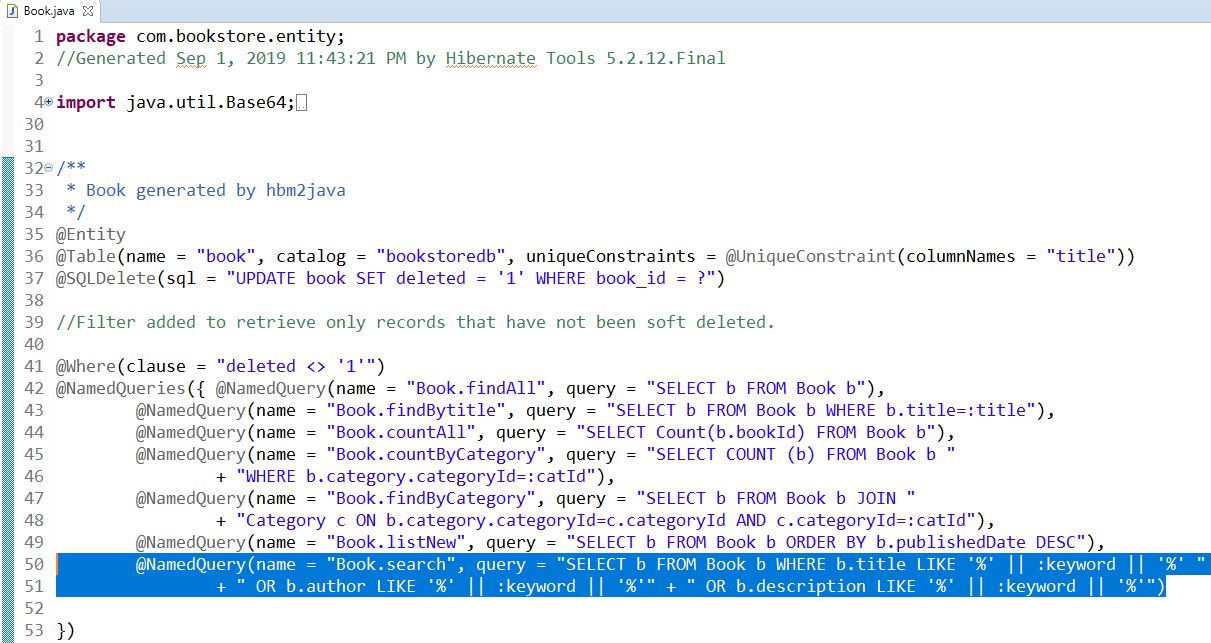
This method returns a list of Book Objects search() and the parameter is keyword and invokes the

findWithNamedQuery() from its superclass(return super.findWithNamedQuery(queryname)).

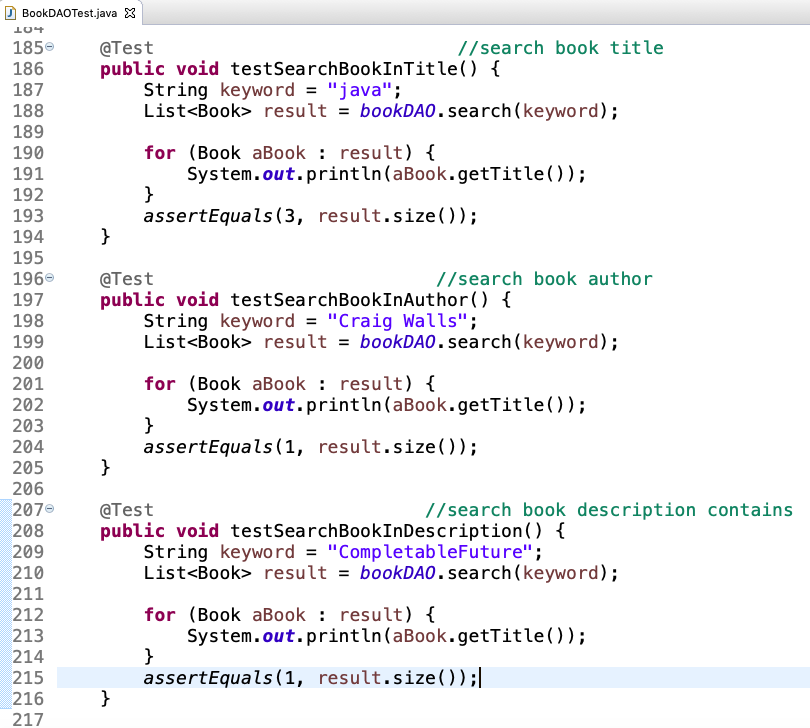
Query name: write the domain model class Book.java (Book.search) and the parameter name in the query is “keyword”, and the value is the keyword string.



**Then, Book domain model class, we create a named query “Book.search”**



**Then, Open BookDAOTest.java class and create test method for search book title, author, description for testing code**



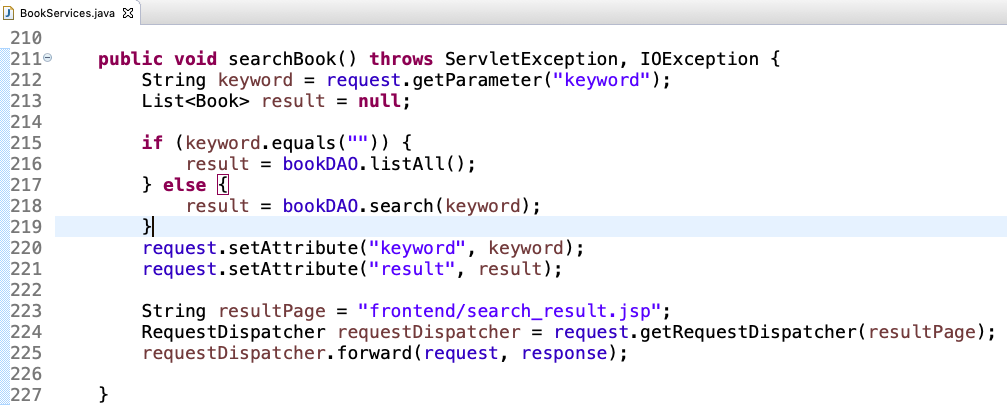
**2.Create SearchBookServlet.java**

In the SearchBookservlet class, it invokes the search()method on the BookServices class so we create an instance of the BookServices class in the SearchBooksevlet class and call the method searchBook().



**3. Call the searchBook() method in BookServices.java**

In this method, we get the keyword from the request.getParameter(“keyword”). If it is black keyword (if(keyword.equal(“”)))then list all books from the BookDAO.listAll() on index.jsp home page on front end side and if it is not blank keyword, we assign the result collection to the result of the search() method for the given keyword and then we set this object result as a request attribute. finally, we forward the request to the search\_result.jsp page.



**4.Code search Result Page(search\_result.jsp)**

In Jsp page, we used JSTL’s length() function so add tag lib directive to use JSTL core tags and function tags <%@ taglib prefix=*"c"* uri=*"*[*http://java.sun.com/jsp/jstl/core*](http://java.sun.com/jsp/jstl/core)*"*%>

< <%@ taglib prefix=*"fn"* uri=*"*[*http://java.sun.com/jsp/jstl/functions*](http://java.sun.com/jsp/jstl/functions)*"*%>

**search\_result.jsp**

|  |
| --- |
| <%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*  pageEncoding=*"ISO-8859-1"*%>  <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>  <%@ taglib prefix=*"fn"* uri=*"http://java.sun.com/jsp/jstl/functions"*%>  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  <html>  <head>  <meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>  <title>Results for "${keyword}" - Online DK Bookstore</title>  <link rel=*"stylesheet"*  href=*"https://fonts.googleapis.com/css?family=Roboto|Open+Sans"*>  <link rel=*"stylesheet"*  href=*"https://maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"*>  <link rel=*"stylesheet"*  href=*"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"*>  <script  src=*"https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"*></script>  <script  src=*"https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"*></script>  <style type=*"text/css"*>  **body** {  background: *#e2eaef*;  font-family: *"Open Sans", sans-serif*;  }  **h2** {  color: *#000*;  font-size: *26px*;  font-weight: *300*;  text-align: *center*;  text-transform: *uppercase,lowercase*;  position: *relative*;  margin: *30px 0 60px*;  }  **h2***::after* {  content: *""*;  width: *100px*;  position: *absolute*;  margin: *0 auto*;  height: *4px*;  border-radius: *1px*;  background: *#7ac400*;  left: *0*;  right: *0*;  bottom: *-20px*;  }  *.carousel* {  margin: *50px auto*;  padding: *0 70px*;  }  *.carousel* *.item* {  color: *#747d89*;  min-height: *325px*;  text-align: *center*;  overflow: *hidden*;  }  *.carousel* *.thumb-wrapper* {  padding: *25px 15px*;  background: *#fff*;  border-radius: *6px*;  text-align: *center*;  position: *relative*;  box-shadow: *0 2px 3px rgba(0, 0, 0, 0.2)*;  }  *.carousel* *.item* *.img-box* {  height: *120px*;  margin-bottom: *20px*;  width: *100%*;  position: *relative*;  }  *.carousel* *.item* **img** {  max-width: *100%*;  max-height: *100%*;  display: *inline-block*;  position: *absolute*;  bottom: *0*;  margin: *0 auto*;  left: *0*;  right: *0*;  }  *.carousel* *.item* **h4** {  font-size: *18px*;  }  *.carousel* *.item* **h4,** *.carousel* *.item* **p,** *.carousel* *.item* **ul** {  margin-bottom: *5px*;  }  *.carousel* *.thumb-content* *.btn* {  color: *#7ac400*;  font-size: *11px*;  text-transform: *uppercase*;  font-weight: *bold*;  background: *none*;  border: *1px solid #7ac400*;  padding: *6px 14px*;  margin-top: *5px*;  line-height: *16px*;  border-radius: *20px*;  }  *.carousel* *.thumb-content* *.btn:hover***,** *.carousel* *.thumb-content* *.btn:focus*  {  color: *#fff*;  background: *#7ac400*;  box-shadow: *none*;  }  *.carousel* *.thumb-content* *.btn* **i** {  font-size: *14px*;  font-weight: *bold*;  margin-left: *5px*;  }  *.carousel* *.carousel-control* {  height: *44px*;  width: *40px*;  background: *#7ac400*;  margin: *auto 0*;  border-radius: *4px*;  opacity: *0.8*;  }  *.carousel* *.carousel-control:hover* {  background: *#78bf00*;  opacity: *1*;  }  *.carousel* *.carousel-control* **i** {  font-size: *36px*;  position: *absolute*;  top: *50%*;  display: *inline-block*;  margin: *-19px 0 0 0*;  z-index: *5*;  left: *0*;  right: *0*;  color: *#fff*;  text-shadow: *none*;  font-weight: *bold*;  }  *.carousel* *.item-price* {  font-size: *13px*;  padding: *2px 0*;  }  *.carousel* *.item-price* **strike** {  opacity: *0.7*;  margin-right: *5px*;  }  *.carousel* *.carousel-control.left* **i** {  margin-left: *-2px*;  }  *.carousel* *.carousel-control.right* **i** {  margin-right: *-4px*;  }  *.carousel* *.carousel-indicators* {  bottom: *-50px*;  }  *.carousel-indicators* **li,** *.carousel-indicators* **li***.active* {  width: *10px*;  height: *10px*;  margin: *4px*;  border-radius: *50%*;  border: *none*;  }  *.carousel-indicators* **li** {  background: *rgba(0, 0, 0, 0.2)*;  }  *.carousel-indicators* **li***.active* {  background: *rgba(0, 0, 0, 0.6)*;  }  *.carousel* *.wish-icon* {  position: *absolute*;  right: *10px*;  top: *10px*;  z-index: *99*;  cursor: *pointer*;  font-size: *16px*;  color: *#abb0b8*;  }  *.carousel* *.wish-icon* *.fa-heart* {  color: *#ff6161*;  }  *.star-rating* **li** {  padding: *0*;  }  *.star-rating* **i** {  font-size: *14px*;  color: *#ffc000*;  }  </style>  </head>  <body>  <jsp:directive.include file=*"header.jsp"* />  <div class=*"form-group"* align=*"center"*>  <c:if test=*"*${fn:length(result)==0}*"*>  <h2>  <b>No Results for "${keyword}"</b>  </h2>  </c:if>  <c:if test=*"*${fn:length(result)>0}*"*>  <div class=*"col-md-12"* align=*"left"*  style="width: *100%*; margin: *0 auto*;">  <h2>  <b>Results for "${keyword}"</b>  </h2>  </div>  <c:forEach items=*"*${result}*"* var=*"book"*>  <div>  <div style="display: *inline-block*; margin: *20px*; width: *9%*"  align=*"left"*>  <div>  <a href=*"view\_book?id=*${book.bookId}*"*> <img  class=*"book-small"*  src=*"data:image/jpg;base64,*${book.base64Image}*"*  class=*"img-responsive img-fluid"* width=*"128"* height=*"164"* />  </a>  </div>  </div>  <div  style="display: *inline-block*; margin: *20px*; vertical-align: *top*; width: *60%*"  align=*"left"*>  <div>  <h4>  <a href=*"view\_book?id=*${book.bookId}*"*> <b>${book.title}</b>  </a>  </h4>  </div>  <div>  <jsp:directive.include file=*"book\_rating.jsp"* />  <a href=*"view\_book?id=*${book.bookId}*"*>(${fn:length(book.reviews)})</a>  </div>  <div>  <i><b>${book.author}</b></i>  </div>  <br />  <div>  <p>${fn:substring(book.description, 0, 100)}...</p>  </div>  </div>  <div  style="display: *inline-block*; margin: *20px*; vertical-align: *top*;">  <h3><b>$${book.price}</b></h3>  <h3>  <a href=*"add\_to\_cart?book\_id=*${book.bookId}*"* class=*"btn btn-success"*><b>Add to Cart</b></a>  </h3>  </div>  </div>  </c:forEach>  </c:if>  </div>  <jsp:directive.include file=*"footer.jsp"* />  </body>  </html> |

**Here I attached full implementation for Book functionality.**

|  |
| --- |
| **com.bookstore.entity** |
| **Book.java** |
| **package** com.bookstore.entity;  // Generated Sep 1, 2019 11:43:21 PM by Hibernate Tools 5.2.12.Final  **import** java.util.Base64;  **import** java.util.Comparator;  **import** java.util.Date;  **import** java.util.HashSet;  **import** java.util.Set;  **import** java.util.TreeSet;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.FetchType;  **import** javax.persistence.GeneratedValue;  **import** **static** javax.persistence.GenerationType.***IDENTITY***;  **import** javax.persistence.Id;  **import** javax.persistence.JoinColumn;  **import** javax.persistence.ManyToOne;  **import** javax.persistence.NamedQueries;  **import** javax.persistence.NamedQuery;  **import** javax.persistence.OneToMany;  **import** javax.persistence.Table;  **import** javax.persistence.Temporal;  **import** javax.persistence.TemporalType;  **import** javax.persistence.Transient;  **import** javax.persistence.UniqueConstraint;  **import** org.hibernate.annotations.SQLDelete;  **import** org.hibernate.annotations.Where;  /\*\*  \* Book generated by hbm2java  \*/  @Entity  @Table(name = "book", catalog = "bookstoredb", uniqueConstraints = @UniqueConstraint(columnNames = "title"))  @SQLDelete(sql = "UPDATE book SET deleted = '1' WHERE book\_id = ?")  //Filter added to retrieve only records that have not been soft deleted.  @Where(clause = "deleted <> '1'")  @NamedQueries({ @NamedQuery(name = "Book.findAll", query = "SELECT b FROM Book b ORDER BY b.title"),  @NamedQuery(name = "Book.findBytitle", query = "SELECT b FROM Book b WHERE b.title=:title"),  @NamedQuery(name = "Book.countAll", query = "SELECT Count(b.bookId) FROM Book b"),  @NamedQuery(name = "Book.countByCategory", query = "SELECT COUNT (b) FROM Book b "  + "WHERE b.category.categoryId=:catId"),  @NamedQuery(name = "Book.findByCategory", query = "SELECT b FROM Book b JOIN "  + "Category c ON b.category.categoryId = c.categoryId AND c.categoryId= :catId"),  @NamedQuery(name = "Book.listNew", query = "SELECT b FROM Book b ORDER BY b.publishedDate DESC"),  @NamedQuery(name = "Book.search", query = "SELECT b FROM Book b WHERE b.title LIKE '%' || :keyword || '%'"  + " OR b.author LIKE '%' || :keyword || '%'" + " OR b.description LIKE '%' || :keyword || '%'")  })  **public** **class** Book **implements** java.io.Serializable {  /\*\*  \*  \*/  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  **private** Integer bookId;  **private** Category category;  **private** String title;  **private** String author;  **private** String description;  **private** String isbn;  **private** **byte**[] image;  **private** String base64Image;  **private** **float** price;  **private** **int** unitstock;  **private** Date publishedDate;  **private** Date lastUpdateTime;  **private** Set<Review> reviews = **new** HashSet<Review>(0);  **private** Set<OrderDetail> orderDetails = **new** HashSet<OrderDetail>(0);  **private** **boolean** deleted;  **public** Book() {  }  **public** Book(Integer bookId) {  **super**();  **this**.bookId = bookId;  }  **public** Book(String title, String author, String description, String isbn, **byte**[] image, **float** price,  Date publishedDate,**int** unitstock, Date lastUpdateTime) {  **this**.title = title;  **this**.author = author;  **this**.description = description;  **this**.isbn = isbn;  **this**.image = image;  **this**.price = price;  **this**.unitstock=unitstock;  **this**.publishedDate = publishedDate;  **this**.lastUpdateTime = lastUpdateTime;  }  **public** Book(Category category, String title, String author, String description, String isbn, **byte**[] image,  **float** price,**int** unitstock, Date publishedDate, Date lastUpdateTime, Set<Review> reviews, Set<OrderDetail> orderDetails) {  **this**.category = category;  **this**.title = title;  **this**.author = author;  **this**.description = description;  **this**.isbn = isbn;  **this**.image = image;  **this**.price = price;  **this**.unitstock=unitstock;  **this**.publishedDate = publishedDate;  **this**.lastUpdateTime = lastUpdateTime;  **this**.reviews = reviews;  **this**.orderDetails = orderDetails;  }  @Id  @GeneratedValue(strategy = ***IDENTITY***)  @Column(name = "book\_id", unique = **true**, nullable = **false**)  **public** Integer getBookId() {  **return** **this**.bookId;  }  **public** **void** setBookId(Integer bookId) {  **this**.bookId = bookId;  }  @Column(name = "deleted", nullable = **false**)  **public** **boolean** isDeleted() {  **return** **this**.deleted;  }  **public** **void** setDeleted(**boolean** deleted) {  **this**.deleted = deleted;  }  @ManyToOne(fetch = FetchType.***EAGER***)  @JoinColumn(name = "category\_id")  **public** Category getCategory() {  **return** **this**.category;  }  **public** **void** setCategory(Category category) {  **this**.category = category;  }  @Column(name = "title", unique = **true**, nullable = **false**, length = 45)  **public** String getTitle() {  **return** **this**.title;  }  **public** **void** setTitle(String title) {  **this**.title = title;  }  @Column(name = "author", nullable = **false**, length = 45)  **public** String getAuthor() {  **return** **this**.author;  }  **public** **void** setAuthor(String author) {  **this**.author = author;  }  @Column(name = "description", nullable = **false**, length = 16777215)  **public** String getDescription() {  **return** **this**.description;  }  **public** **void** setDescription(String description) {  **this**.description = description;  }  @Column(name = "isbn", nullable = **false**, length = 45)  **public** String getIsbn() {  **return** **this**.isbn;  }  **public** **void** setIsbn(String isbn) {  **this**.isbn = isbn;  }  @Column(name = "image", nullable = **false**)  **public** **byte**[] getImage() {  **return** **this**.image;  }  **public** **void** setImage(**byte**[] image) {  **this**.image = image;  }  @Column(name = "price", nullable = **false**, precision = 12, scale = 0)  **public** **float** getPrice() {  **return** **this**.price;  }    **public** **void** setPrice(**float** price) {  **this**.price = price;  }    @Column(name = "unitstock", nullable = **false**)  **public** **int** getUnitstock() {  **return** **this**.unitstock;  }  **public** **void** setUnitstock(**int** unitstock) {  **this**.unitstock = unitstock;  }    @Temporal(TemporalType.***DATE***)  @Column(name = "published\_date", nullable = **false**, length = 10)  **public** Date getPublishedDate() {  **return** **this**.publishedDate;  }  **public** **void** setPublishedDate(Date publishedDate) {  **this**.publishedDate = publishedDate;  }  @Temporal(TemporalType.***TIMESTAMP***)  @Column(name = "last\_update\_time", nullable = **false**, length = 19)  **public** Date getLastUpdateTime() {  **return** **this**.lastUpdateTime;  }  **public** **void** setLastUpdateTime(Date lastUpdateTime) {  **this**.lastUpdateTime = lastUpdateTime;  }  @OneToMany(fetch = FetchType.***EAGER***, mappedBy = "book")  **public** Set<Review> getReviews() { //this methods correct order  TreeSet<Review> sortedReviews=**new** TreeSet<Review>(**new** Comparator<Review>() {  @Override  **public** **int** compare(Review review1, Review review2) {  **return** review2.getReviewTime().compareTo(review1.getReviewTime());  }  });  sortedReviews.addAll(reviews);  **return** **this**.reviews;  }  **public** **void** setReviews(Set<Review> reviews) {  **this**.reviews = reviews;  }  @OneToMany(fetch = FetchType.***LAZY***, mappedBy = "book")  **public** Set<OrderDetail> getOrderDetails() {  **return** **this**.orderDetails;  }  **public** **void** setOrderDetails(Set<OrderDetail> orderDetails) {  **this**.orderDetails = orderDetails;  }  @Transient  **public** String getBase64Image() {  **this**.base64Image = Base64.*getEncoder*().encodeToString(**this**.image);  **return** **this**.base64Image;  }  @Transient  **public** **void** setBase64Image(String base64Image) {  **this**.base64Image = base64Image;  }  @Transient // Average rating  **public** **float** getAverageRating() {  **float** averageRating = 0.0f;  **float** sum = 0.0f;  **if** (reviews.isEmpty()) {  **return** 0.0f;  }  **for** (Review review : reviews) {  sum += review.getRating();  }  averageRating = sum / reviews.size();  **return** averageRating;  }  @Transient  **public** String getRatingStars() {  **float** averageRating = getAverageRating();    **return** getRatingString(averageRating);  }  @Transient  **public** String getRatingString(**float** averageRating) {  String result = "";  **int** numberOfStarOn = (**int**) averageRating;  **for** (**int** i = 1; i <= numberOfStarOn; i++) {  result += "on,";  }  **int** next = numberOfStarOn + 1;  **if** (averageRating > numberOfStarOn) {  result += "half,";  next++;  }  **for** (**int** j = next; j <= 5; j++) {  result += "off,";  }  **return** result.substring(0, result.length() - 1);  }  @Override  **public** **int** hashCode() {  **final** **int** prime = 31;  **int** result = 1;  result = prime \* result + ((bookId == **null**) ? 0 : bookId.hashCode());  **return** result;  }  @Override // hash code moethod  **public** **boolean** equals(Object obj) {  **if** (**this** == obj)  **return** **true**;  **if** (obj == **null**)  **return** **false**;  **if** (getClass() != obj.getClass())  **return** **false**;  Book other = (Book) obj;  **if** (bookId == **null**) {  **if** (other.bookId != **null**)  **return** **false**;  } **else** **if** (!bookId.equals(other.bookId))  **return** **false**;  **return** **true**;  }  } |

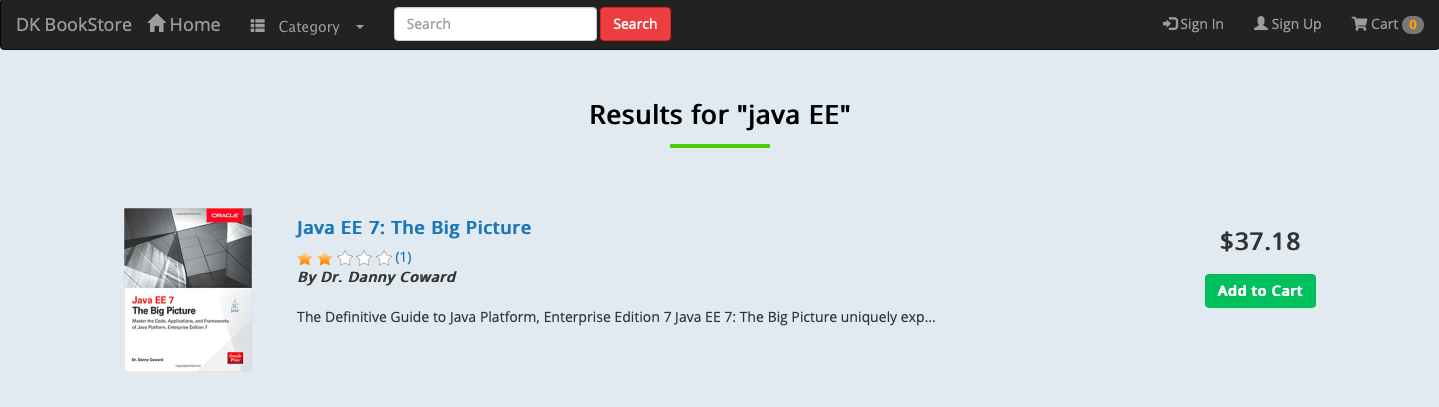
|  |
| --- |
| **com.bookstore.dao** |
| **GenericDAO.java (Interface class)** |
| **package** com.bookstore.dao;  **import** java.util.List;  //GenericDAO interface define operations that  //are common to all specific DAO classes  **public** **interface** GenericDAO<T> {  **public** T create(T t);  **public** T update(T t);  **public** T get(Object id);  **public** **void** delete(Object id);  **public** List<T> listAll();  **public** **long** count(); // this method returns total  //number of entity and total number of rows in a table  } |
|  |
| **JpaDAO.java** |
| **package** com.bookstore.dao;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Map.Entry;  **import** java.util.Set;  **import** javax.persistence.EntityManager;  **import** javax.persistence.EntityManagerFactory;  **import** javax.persistence.Persistence;  **import** javax.persistence.Query;  //This class JPA to implement some persistence operations  //that are common to all specific DAO classes  //All DAO class have subclasses of JpaDAO class.  //All DAO class implementing the Generic interface.  //<E> is Parameterize type of generic type E  //implement all operation for all subclasses  **public** **class** JpaDAO<E> {  **private** **static** EntityManagerFactory *entityManagerFactory*;  **protected** EntityManager entityManager;  **static** {  *entityManagerFactory* = Persistence.*createEntityManagerFactory*("OnlineBookStoreWebsite");  }  **public** JpaDAO() {  }  **public** E create(E entity) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  entityManager.getTransaction().begin();  entityManager.persist(entity);  entityManager.flush();  entityManager.refresh(entity);  entityManager.getTransaction().commit();  entityManager.close();  **return** entity;  }  **public** E update(E entity) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  entityManager.getTransaction().begin();  entityManager.merge(entity);  entityManager.getTransaction().commit();  entityManager.close();  **return** entity;  }  **public** E find(Class<E> type, Object id) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  E entity = entityManager.find(type, id);  **if** (entity != **null**) {  entityManager.refresh(entity);  }  entityManager.close();  **return** entity;  }  **public** **void** delete(Class<E> type, Object id) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  entityManager.getTransaction().begin();  Object reference = entityManager.getReference(type, id);  entityManager.remove(reference);  entityManager.getTransaction().commit();  entityManager.close();  }  **public** List<E> findWithNamedQuery(String queryName) {  // this method returns a list of entity objects  EntityManager entityManager = *entityManagerFactory*.createEntityManager();    Query query = entityManager.createNamedQuery(queryName); // create query object from the entity manager  List<E> result = query.getResultList(); // returns results    entityManager.close();  **return** result;  }  **public** List<E> findWithNamedQuery(String queryName, String paramName, Object paramValue) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(queryName);  query.setParameter(paramName, paramValue);  List<E> result = query.getResultList();  entityManager.close();  **return** result;  }  **public** List<E> findWithNamedQuery(String queryName, Map<String, Object> parameters) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(queryName);  Set<Entry<String, Object>> setParameters = parameters.entrySet();  **for** (Entry<String, Object> entry : setParameters) {  query.setParameter(entry.getKey(), entry.getValue());  }  List<E> result = query.getResultList();  entityManager.close();  **return** result;  }  // list new book  **public** List<E> findWithNamedQuery(String queryName, **int** firstResult, **int** maxResult) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(queryName);  query.setFirstResult(firstResult);  query.setMaxResults(maxResult);  List<E> result = query.getResultList();  entityManager.close();  **return** result;  }    // list new book  **public** List<Object[]> findWithNamedQueryObjects(String queryName, **int** firstResult, **int** maxResult) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(queryName);  query.setFirstResult(firstResult);  query.setMaxResults(maxResult);  List<Object[]> result = query.getResultList();  entityManager.close();  **return** result;  }  **public** **long** countWithNamedQuery(String QueryName) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(QueryName);  **long** result = (**long**) query.getSingleResult();  entityManager.close();  **return** result;  }  **public** **long** countWithNamedQuery(String QueryName, String paramName, Object paramValue) {  EntityManager entityManager = *entityManagerFactory*.createEntityManager();  Query query = entityManager.createNamedQuery(QueryName);  query.setParameter(paramName, paramValue);  **long** result = (**long**) query.getSingleResult();  entityManager.close();  **return** result;  }  **public** **void** close() {  **if** (*entityManagerFactory* != **null**) {  *entityManagerFactory*.close();  }  }  } |
|  |
| **BookDAO.java** |
| **package** com.bookstore.dao;  **import** java.util.ArrayList;  **import** java.util.Date;  **import** java.util.List;  **import** com.bookstore.entity.Book;  **public** **class** BookDAO **extends** JpaDAO<Book> **implements** GenericDAO<Book> {  **public** BookDAO() {  }  @Override  **public** Book create(Book book) {  book.setLastUpdateTime(**new** Date());  **return** **super**.create(book);  }  @Override  **public** Book update(Book book) {  book.setLastUpdateTime(**new** Date());  **return** **super**.update(book);  }  @Override  **public** Book get(Object bookId) {  **return** **super**.find(Book.**class**, bookId);  }  @Override  **public** **void** delete(Object bookId) {  **super**.delete(Book.**class**, bookId);  }  @Override  **public** List<Book> listAll() {  **return** **super**.findWithNamedQuery("Book.findAll");  }  **public** Book findByTitle(String title) {  List<Book> result = **super**.findWithNamedQuery("Book.findBytitle", "title", title);  **if** (!result.isEmpty()) {  **return** result.get(0);  }  **return** **null**;  }  **public** List<Book> listByCategory(**int** categoryId) {  **return** **super**.findWithNamedQuery("Book.findByCategory", "catId", categoryId);  }  **public** List<Book> listNewBooks() {  **return** **super**.findWithNamedQuery("Book.listNew", 0, 4);  }  **public** List<Book> search(String keyword) {  **return** **super**.findWithNamedQuery("Book.search", "keyword", keyword);  }  @Override  **public** **long** count() {  **return** **super**.countWithNamedQuery("Book.countAll");  }  **public** **long** countByCategory(**int** categoryId) {  **return** **super**.countWithNamedQuery("Book.countByCategory", "catId", categoryId);  }  **public** List<Book> listBestSellingBooks() {  **return** **super**.findWithNamedQuery("OrderDetail.bestSelling", 0, 4);  }  **public** List<Book> listMostFavoredBooks() {  List<Book> mostFavoredBooks = **new** ArrayList<>();  List<Object[]> result = **super**.findWithNamedQueryObjects("Review.mostFavoredBooks", 0, 4);  **if** (!result.isEmpty()) {  **for** (Object[] elements : result) {  Book book = (Book) elements[0];  mostFavoredBooks.add(book);  }  }  **return** mostFavoredBooks;  }  } |

|  |
| --- |
| **com.bookstore.services** |
| **BookServices.java** |
| **package** com.bookstore.service;  **import** java.io.IOException;  **import** java.io.InputStream;  **import** java.text.DateFormat;  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** java.util.List;  **import** javax.servlet.RequestDispatcher;  **import** javax.servlet.ServletException;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** javax.servlet.http.Part;  **import** com.bookstore.dao.BookDAO;  **import** com.bookstore.dao.CategoryDAO;  **import** com.bookstore.entity.Book;  **import** com.bookstore.entity.Category;  **public** **class** BookServices {  **private** BookDAO bookDAO;  **private** CategoryDAO categoryDAO;  **private** HttpServletRequest request;  **private** HttpServletResponse response;  **public** BookServices(HttpServletRequest request, HttpServletResponse response) {  **super**();  **this**.request = request;  **this**.response = response;  bookDAO = **new** BookDAO();  categoryDAO = **new** CategoryDAO();  }  **public** **void** listBooks() **throws** ServletException, IOException {  listBooks(**null**);  }  **public** **void** listBooks(String message) **throws** ServletException, IOException {  List<Book> listBooks = bookDAO.listAll();  request.setAttribute("listBooks", listBooks);  **if** (message != **null**) {  request.setAttribute("message", message);  }  String listPage = "book\_list.jsp";  RequestDispatcher requestDispatcher = request.getRequestDispatcher(listPage);  requestDispatcher.forward(request, response);  }  **public** **void** showBookNewForm() **throws** ServletException, IOException {  List<Category> listCategory = categoryDAO.listAll();  request.setAttribute("listCategory", listCategory);  String newPage = "book\_form.jsp";  RequestDispatcher requestDispatcher = request.getRequestDispatcher(newPage);  requestDispatcher.forward(request, response);  }  **public** **void** createBook() **throws** ServletException, IOException {  String title = request.getParameter("title");  Book existBook = bookDAO.findByTitle(title);  **if** (existBook != **null**) {  String message = "Could not create new book because the title '" + title + "' already exists.";  listBooks(message);  **return**;  }  Book newBook = **new** Book();  readBookFields(newBook);  Book createdBook = bookDAO.create(newBook);  **if** (createdBook.getBookId() > 0) {  String message = "A new book has been created successfully.";  listBooks(message);  }  }  **public** **void** readBookFields(Book book) **throws** ServletException, IOException {  String title = request.getParameter("title");  String author = request.getParameter("author");  String description = request.getParameter("description");  String isbn = request.getParameter("isbn");  **float** price = Float.*parseFloat*(request.getParameter("price"));  Integer unitstock=Integer.*parseInt*(request.getParameter("stockAvailable"));  DateFormat dateFormat = **new** SimpleDateFormat("MM/dd/yyyy");  Date publishDate = **null**;  **try** {  publishDate = dateFormat.parse(request.getParameter("publishedDate"));  } **catch** (ParseException ex) {  ex.printStackTrace();  **throw** **new** ServletException("Error while parsing publish date(format is MM/dd/yyyy)");  }  book.setTitle(title);  book.setAuthor(author);  book.setDescription(description);  book.setIsbn(isbn);  book.setPublishedDate(publishDate);    Integer categoryId = Integer.*parseInt*(request.getParameter("category"));  Category category = categoryDAO.get(categoryId);  book.setCategory(category);  book.setPrice(price);  book.setUnitstock(unitstock);    Part part = request.getPart("bookImage");  **if** (part != **null** && part.getSize() > 0) {  **long** size = part.getSize();  **byte**[] imageBytes = **new** **byte**[(**int**) size];  InputStream inputStream = part.getInputStream();  inputStream.read(imageBytes);  inputStream.close();  book.setImage(imageBytes);  }  }  **public** **void** editBook() **throws** ServletException, IOException {  Integer bookId = Integer.*parseInt*(request.getParameter("id"));  Book book = bookDAO.get(bookId);  String editPage = "book\_form.jsp";  **if** (book != **null**) {  List<Category> listCategory = categoryDAO.listAll();  request.setAttribute("book", book);  request.setAttribute("listCategory", listCategory);  } **else** {  editPage = "message.jsp";  String errorMessage = "Could not find book with ID " + bookId;  request.setAttribute("message", errorMessage);  }  RequestDispatcher requestDispatcher = request.getRequestDispatcher(editPage);  requestDispatcher.forward(request, response);  }  **public** **void** updateBook() **throws** ServletException, IOException {  Integer bookId = Integer.*parseInt*(request.getParameter("bookId"));  String title = request.getParameter("title");  Book exitBook = bookDAO.get(bookId);  Book bookByTitle = bookDAO.findByTitle(title);  **if** (bookByTitle != **null** && !exitBook.equals(bookByTitle)) {  String message = "Could not update book because there's another book having same title!";  listBooks(message);  **return**;  }  readBookFields(exitBook);  bookDAO.update(exitBook);  String message = "The book has been updated successfully!";  listBooks(message);  }  **public** **void** deleteBook() **throws** ServletException, IOException {  Integer bookId = Integer.*parseInt*(request.getParameter("id"));  bookDAO.delete(bookId);  String message = "The Book with ID " + bookId  + " has been soft deleted successfully from here but store in database!";  listBooks(message);  }  **public** **void** listBooksCategory() **throws** ServletException, IOException {  Integer categoryId = Integer.*parseInt*(request.getParameter("id"));  List<Book> listBooks = bookDAO.listByCategory(categoryId);  Category category = categoryDAO.get(categoryId);  List<Category> listCategory = categoryDAO.listAll();  request.setAttribute("listCategory", listCategory);  request.setAttribute("listBooks", listBooks);  request.setAttribute("category", category);  String listPage = "frontend/books\_list\_by\_Category.jsp";  RequestDispatcher requestDispatcher = request.getRequestDispatcher(listPage);  requestDispatcher.forward(request, response);  }  **public** **void** viewBookDetail() **throws** ServletException, IOException {  Integer bookId = Integer.*parseInt*(request.getParameter("id"));  Book book = bookDAO.get(bookId);  List<Category> listCategory = categoryDAO.listAll();  request.setAttribute("listCategory", listCategory);  request.setAttribute("book", book);  String detailPage = "frontend/book\_detail.jsp";  RequestDispatcher requestDispatcher = request.getRequestDispatcher(detailPage);  requestDispatcher.forward(request, response);  }  **public** **void** searchBook() **throws** ServletException, IOException {  String keyword = request.getParameter("keyword");  List<Book> result = **null**;  **if** (keyword.equals("")) {  result = bookDAO.listAll();  } **else** {  result = bookDAO.search(keyword);  }  request.setAttribute("keyword", keyword);  request.setAttribute("result", result);  String resultPage = "frontend/search\_result.jsp";  RequestDispatcher requestDispatcher = request.getRequestDispatcher(resultPage);  requestDispatcher.forward(request, response);  }  } |

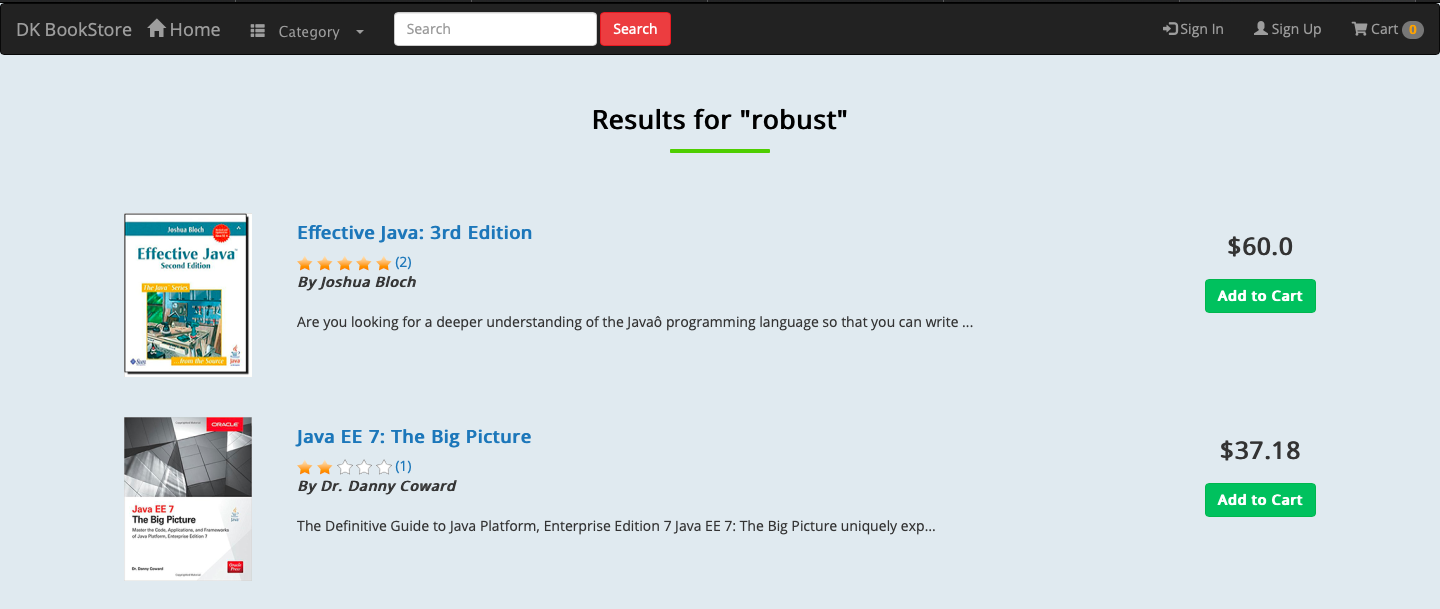
|  |
| --- |
| **com.bookstore.dao //in test folder** |
| **BookDAOTest.java** |
| **package** com.bookstore.dao;  **import** **static** org.junit.Assert.\*;  **import** java.io.IOException;  **import** java.nio.file.Files;  **import** java.nio.file.Paths;  **import** java.text.DateFormat;  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.util.Date;  **import** java.util.List;  **import** javax.persistence.EntityNotFoundException;  **import** org.junit.AfterClass;  **import** org.junit.BeforeClass;  **import** org.junit.Test;  **import** com.bookstore.entity.Book;  **import** com.bookstore.entity.Category;  **public** **class** BookDAOTest {  **private** **static** BookDAO *bookDAO*;  @BeforeClass  **public** **static** **void** setUpBeforeClass() **throws** Exception {  *bookDAO* = **new** BookDAO();  }  @Test  **public** **void** testCreateBook() **throws** ParseException, IOException {  Book newBook = **new** Book();  Category category = **new** Category("java");  category.setCategoryId(1);  newBook.setCategory(category);  newBook.setTitle("Effective Java(3rd Edition)");  newBook.setAuthor("Joshua Bloch");  newBook.setDescription("New Coverage-of generics,enums,annotations,autoboxing");  newBook.setPrice(38.87f);  newBook.setUnitstock(12);  newBook.setIsbn("0-13-468599-7");  DateFormat dateFormate = **new** SimpleDateFormat("MM/dd/yyyy");  Date publishDate = dateFormate.parse("05/28/2008");  newBook.setPublishedDate(publishDate);  String imagePath = "./books/Effective Java.jpg";  **byte**[] imageBytes = Files.*readAllBytes*(Paths.*get*(imagePath));  newBook.setImage(imageBytes);  Book createBook = *bookDAO*.create(newBook);  *assertTrue*(createBook.getBookId() > 0);  }  @Test  **public** **void** testCreate2Book() **throws** ParseException, IOException {  Book newBook = **new** Book();  Category category = **new** Category("java");  category.setCategoryId(1);  newBook.setCategory(category);  newBook.setTitle("Java 8 in Action");  newBook.setAuthor("Raoul-Gabriel Urma");  newBook.setDescription("java 8 in Action is a clearly written guide to the new features of Java 8");  newBook.setPrice(36.72f);  newBook.setIsbn("1-61-729199-4");  DateFormat dateFormate = **new** SimpleDateFormat("MM/dd/yyyy");  Date publishDate = dateFormate.parse("08/28/2014");  newBook.setPublishedDate(publishDate);  String imagePath = "./books/Java 8 in Action.jpg";  **byte**[] imageBytes = Files.*readAllBytes*(Paths.*get*(imagePath));  newBook.setImage(imageBytes);  Book createBook = *bookDAO*.create(newBook);  *assertTrue*(createBook.getBookId() > 0);  }  @Test  **public** **void** testUpdateBook() **throws** ParseException, IOException {  Book exitBook = **new** Book();  exitBook.setBookId(4);  Category category = **new** Category("Java Programming");  category.setCategoryId(1);  exitBook.setCategory(category);  exitBook.setTitle("Effective Java(3rd Edition)");  exitBook.setAuthor("Joshua Bloch");  exitBook.setDescription("New Coverage-of generics,enums,annotations,autoboxing");  exitBook.setPrice(40f);  exitBook.setIsbn("0-13-468599-8");  DateFormat dateFormate = **new** SimpleDateFormat("MM/dd/yyyy");  Date publishDate = dateFormate.parse("05/28/2008");  exitBook.setPublishedDate(publishDate);  String imagePath = "./books/Effective Java.jpg";  **byte**[] imageBytes = Files.*readAllBytes*(Paths.*get*(imagePath));  exitBook.setImage(imageBytes);  Book updatedBook = *bookDAO*.update(exitBook);  *assertEquals*(updatedBook.getTitle(), "Effective Java(3rd Edition)");  }  @Test(expected = EntityNotFoundException.**class**)  **public** **void** testDeleteBookFail() {  Integer bookId = 100;  *bookDAO*.delete(bookId);  }  @Test  **public** **void** testGetBookFail() {  Integer bookId = 99;  Book book = *bookDAO*.get(bookId);  *assertNull*(book);  }  @Test  **public** **void** testGetBookSuccess() {  Integer bookId = 4;  Book book = *bookDAO*.get(bookId);  *assertNotNull*(book);  }  @Test //list all book  **public** **void** testListAll() {  List<Book> listBooks = *bookDAO*.listAll();  **for** (Book aBook : listBooks) {  System.***out***.println(aBook.getTitle() + "-" + aBook.getAuthor());  }  *assertFalse*(listBooks.isEmpty());  }  @Test  **public** **void** testCount() {  **long** totalBooks = *bookDAO*.count();  System.***out***.println("Total Book Count: " + totalBooks);  *assertEquals*(2, totalBooks);  }  @Test  **public** **void** testDeleteBookSuccess() {  Integer bookId = 10;  *bookDAO*.delete(bookId);  *assertTrue*(**true**);  }  @Test  **public** **void** testFindByTitleNotExist() {  String title = "Thinking in java";  Book book = *bookDAO*.findByTitle(title);  *assertNull*(book);  }  @Test  **public** **void** testFindByTitleExist() {  String title = "Effective Java(3rd Edition)";  Book book = *bookDAO*.findByTitle(title);  System.***out***.println("Book Author:" + book.getAuthor());  System.***out***.println("Book Price" + book.getPrice());  *assertNotNull*(book);  }  @Test  **public** **void** testListByCategory() {  **int** categoryId = 1;  List<Book> listBooks = *bookDAO*.listByCategory(categoryId);  *assertTrue*(listBooks.size() > 0);  }  @Test //search book title  **public** **void** testSearchBookInTitle() {  String keyword = "java";  List<Book> result = *bookDAO*.search(keyword);  **for** (Book aBook : result) {  System.***out***.println(aBook.getTitle());  }  *assertEquals*(3, result.size());  }  @Test //search book author  **public** **void** testSearchBookInAuthor() {  String keyword = "Craig Walls";  List<Book> result = *bookDAO*.search(keyword);  **for** (Book aBook : result) {  System.***out***.println(aBook.getTitle());  }  *assertEquals*(1, result.size());  }  @Test //search book description contains  **public** **void** testSearchBookInDescription() {  String keyword = "CompletableFuture";  List<Book> result = *bookDAO*.search(keyword);  **for** (Book aBook : result) {  System.***out***.println(aBook.getTitle());  }  *assertEquals*(1, result.size());  }  @Test  **public** **void** testCountByCategory() {  **int** categoryId = 69;  **long** numOfBooks = *bookDAO*.countByCategory(categoryId);  *assertTrue*(numOfBooks == 7);  }  @Test // list new Books  **public** **void** testListByNewBooks() {  List<Book> listNewBooks = *bookDAO*.listNewBooks();  **for** (Book aBook : listNewBooks) {  System.***out***.println(aBook.getTitle() + "-" + aBook.getPublishedDate());  }  *assertEquals*(4, listNewBooks.size());  }  @Test // list best selling books  **public** **void** testListBestSellingBooks() {  List<Book> topBestSellingBooks = *bookDAO*.listBestSellingBooks();  **for** (Book book : topBestSellingBooks) {  System.***out***.println(book.getTitle());  }  *assertEquals*(4, topBestSellingBooks.size());  }  @Test // list Most Favourite book  **public** **void** testListMostFavoredBooks() {  List<Book> topFavoredBooks = *bookDAO*.listMostFavoredBooks();  **for** (Book book : topFavoredBooks) {  System.***out***.println(book.getTitle());  }  *assertEquals*(4, topFavoredBooks.size());  }  @AfterClass // After all test methods in the test After class  **public** **static** **void** tearDownAfterClass() {  *bookDAO*.close();  }  } |

**Output:**

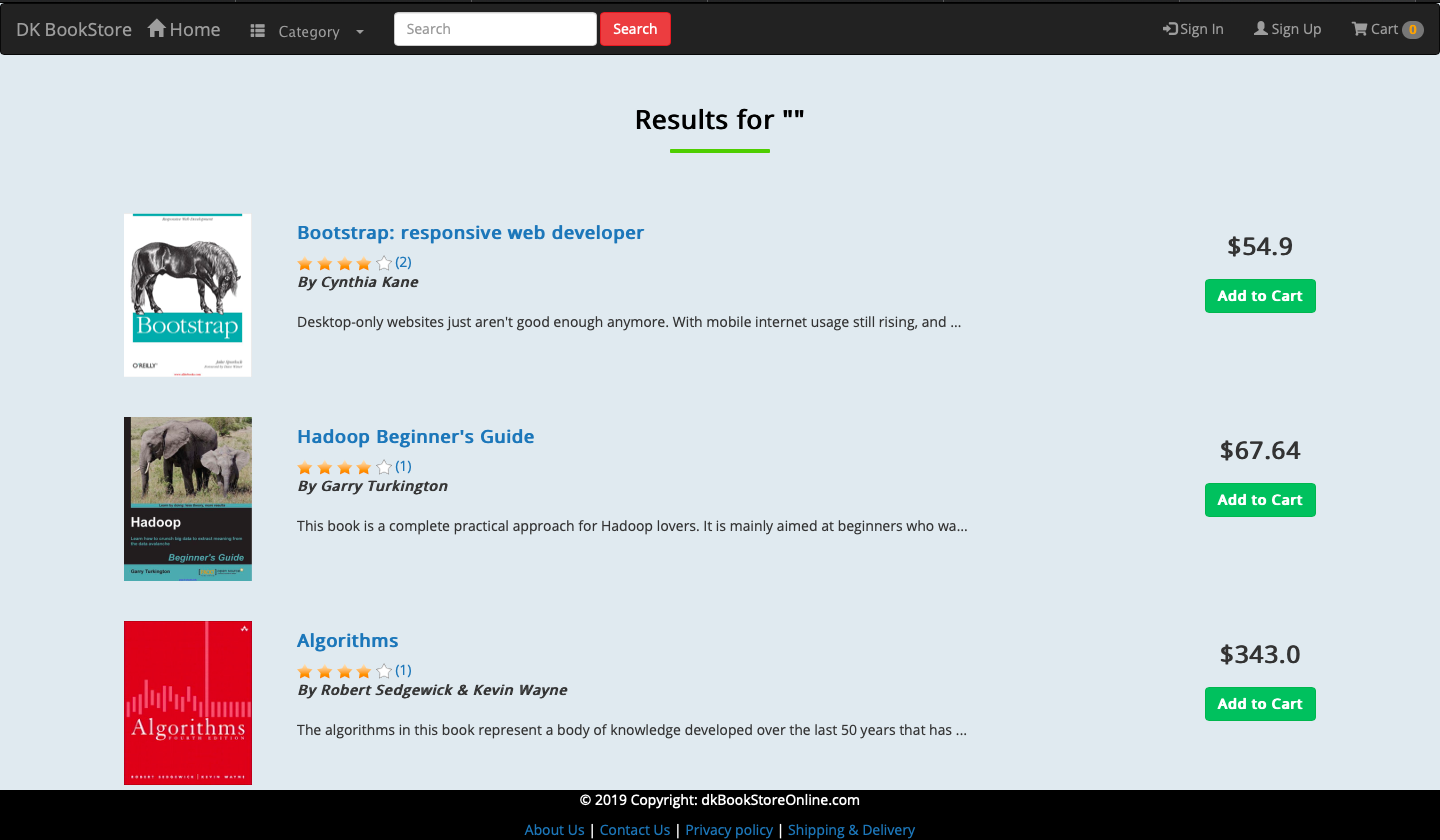
1. search **“java EE”** keyword.

* + ****

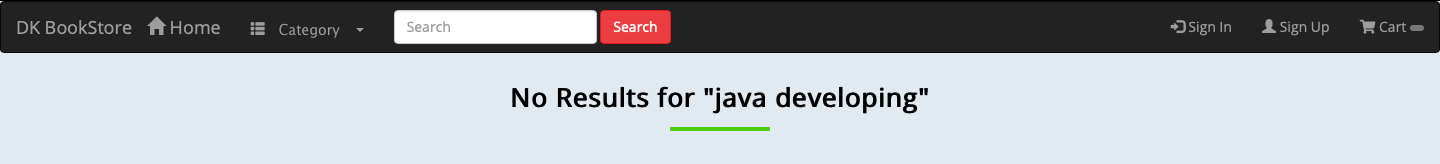
2. Search **“robust”** keyword (Description contains).



3.Search **“”** without keyword.



4. Search **“java develoing”** keyword (no keyword result found).



-----------------------------------------------------------**End**------------------------------------------------------------