**Program Title:** Book Store Management System

**Name:**

**Student Number:**

**Due Date:**

**Short description:**

Book Store Management System – is a program designed for managing a book store. Available books in the store is added to the system with price and quantity. These books can be sold to the customers by adding to the cart and later confirming by purchase. Total and average cost per book is mentioned. All books available in the store can be displayed with price and quantity. All books sold to the customer is stored in register and can be displayed with total and average cost per book.

**Program Specification:**

Book Store Management System – is a program designed for managing a book store.

Initially, there will be no books stored in the system. To add books to the system, book name, available quantity and price per book need to be entered in the bookNameField, quantityField and priceField respectively. Then, Add Books button need to be clicked to complete adding a book to the system. If any of the mentioned fields are missing values or entered format is wrong (i.e. entering string/double in quantityField etc.) then an error dialogue will open with appropriate message and after that some input fields will be restored and focus will be set on the field that had the error (i.e. if name entered correctly but quantity is entered wrong then after error message quantityField will be restored and focus will be set on quantityField). If book is added successfully, book name, available quantity and price per book will be displayed with appropriate header and success message in the textArea.

Secondly, if there are books available in the system, all available book name, available quantity and price per book can be displayed in the textArea by clicking Display All Books button. If no book is available in the system, then clicking the button will display an error dialogue with “No book available” message.

Thirdly, to add book to the cart, book name, available quantity and customer name need to be entered in the bookNameField, quantityField and customerNameField respectively. Then, Add to cart button need to be clicked to complete adding a book to the cart. Books can be searched case insensitively. Like previously, invalid or missing input will cause displaying error dialogue and input field to be restored. For example: If a book is not available in the system or if it is available but asking quantity is greater than available quantity will result in error. For successful input, book will be added to the cart and customer name, book name, quantity and price per book and success message will be displayed in the textArea. Please note that adding books to the cart does not mean books are purchased by the customer. Before purchasing books, all the books need to be added to the cart by the same customer name, concurrently. if customer name is edited before purchasing, all the books added to the cart will be restored.

To purchase all the books added to the cart Purchase button need to be clicked right after all the books are successfully added to the cart. If any other operation like: Add Books, Display All Books, Display Register is occurred before purchasing, all the books added to the cart will be restored. Therefore, all books need to be added to the cart from the beginning. If customer name is edited before purchasing it will result in displaying an error message that customer name is invalid. To successfully purchase, correct customer name needs to be entered or cart need to restored all over from the beginning. After successfully purchase, a receipt including customer name, book name, purchased quantity, price per book, total charge (purchased quantity \* price per book) of all the purchased books and total cost of all books and average cost per book by the customer will be displayed in the textArea. Available books quantity will get reduced after purchasing these. For example: If book store has 10 Harry Potter books. If Jack buys 4 then available quantity of Harry Potter will be 6. If Jones by the rest of 6 books then there will be no Harry Potter books in the store.

Lastly, to see the register of the book store Display Register button needs to be pressed. If no book is sold yet, error dialogue will be displayed with register is empty message. Else, all the books sold will be displayed one after one in the text area with book name, purchased quantity, price per book, total charge and lastly total cost of all books and average cost per book sold by the book store.

Exiting the program will display a thank you message.

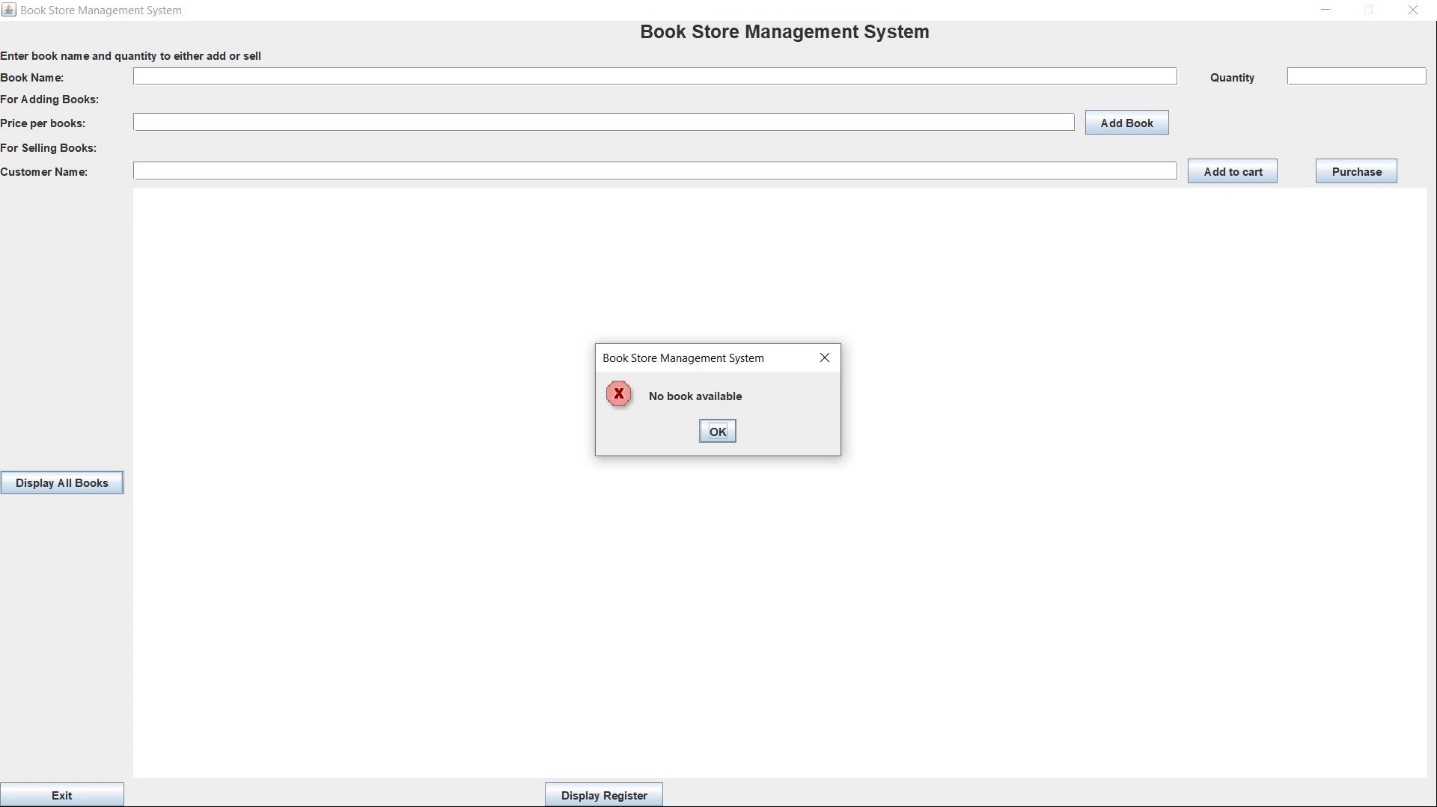
**N.B:**

1. GUI of the program is designed in Intellij IDE. BookStoreManagementSystem.form has the form design and BookStoreManagementSystem.java has the source code of the GUI Form. Running the program in any other IDE may distort the GUI.

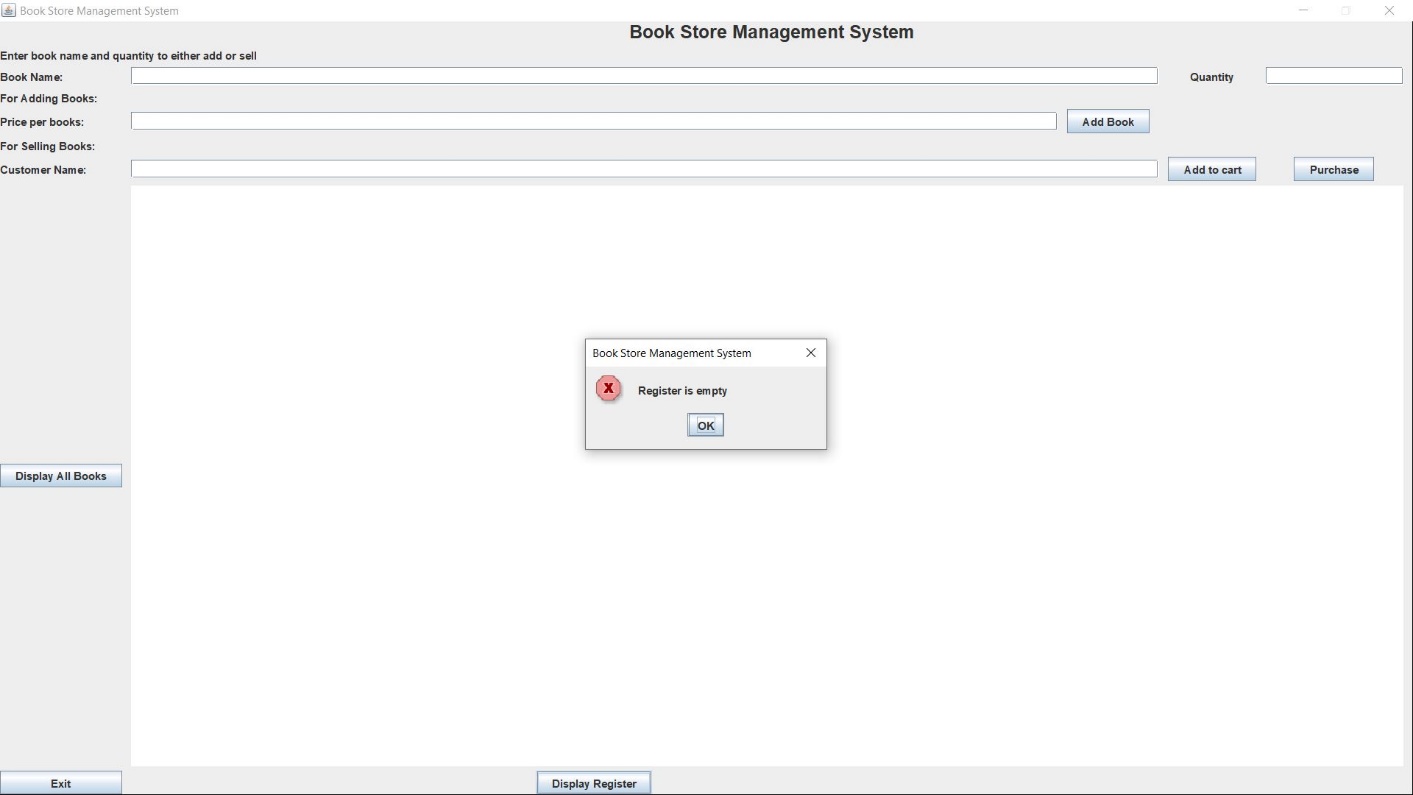
2. bookNameField and quantityField are used for both Add Book and Add to cart operation. For example: for adding books to the system, book name, quantity and price will be taken, customerNameField will be ignored. Also, for adding books to the cart, book name, quantity and customer name will be taken, priceField will be ignored.

3. All the necessary code to run program is stored in BookStoreManagementSystem.java class. Book.java class is created to store book information like book name, available quantity and price per book. Customer class is used to store customer name and book information like book name, purchased quantity and price per book. ArrayList data structure is used to store available books in the store, added to cart books and customer purchase information.

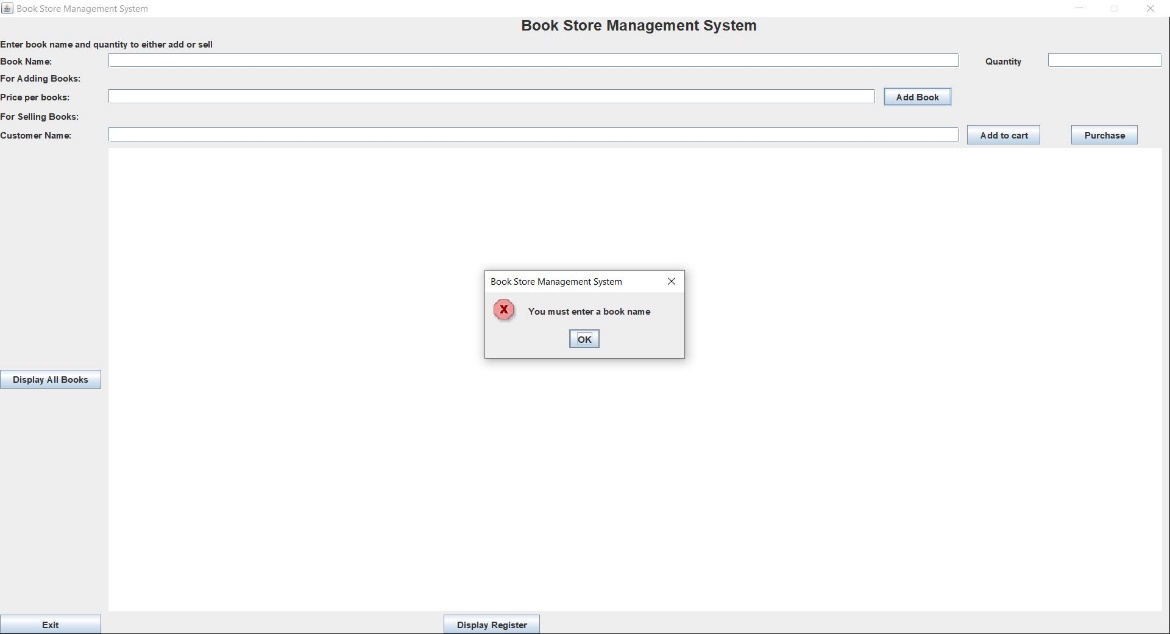
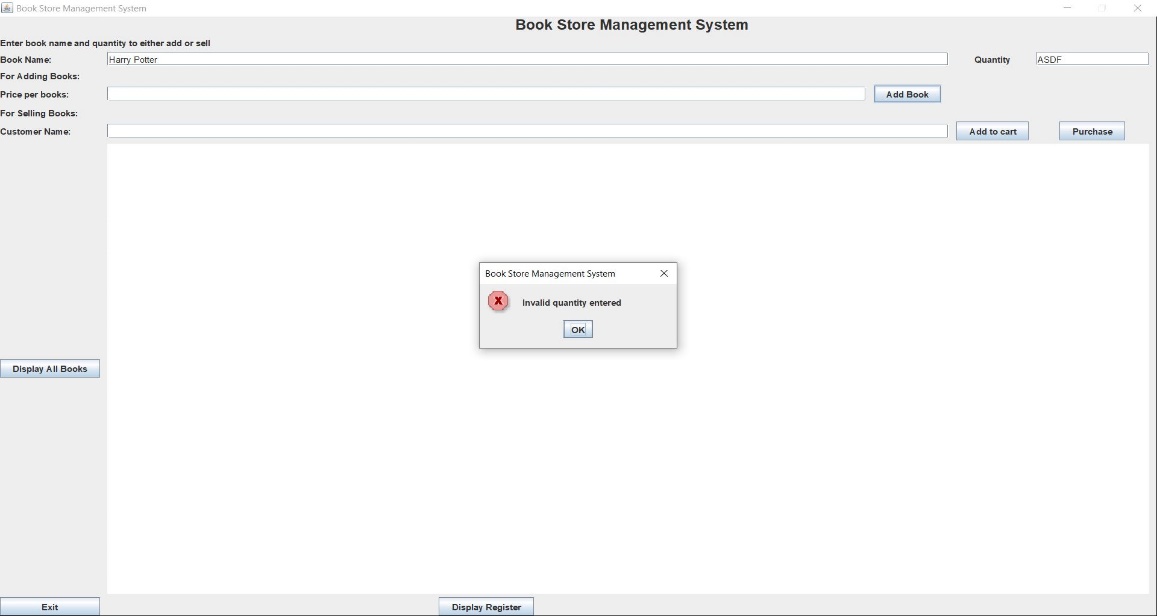
**Screenshots**

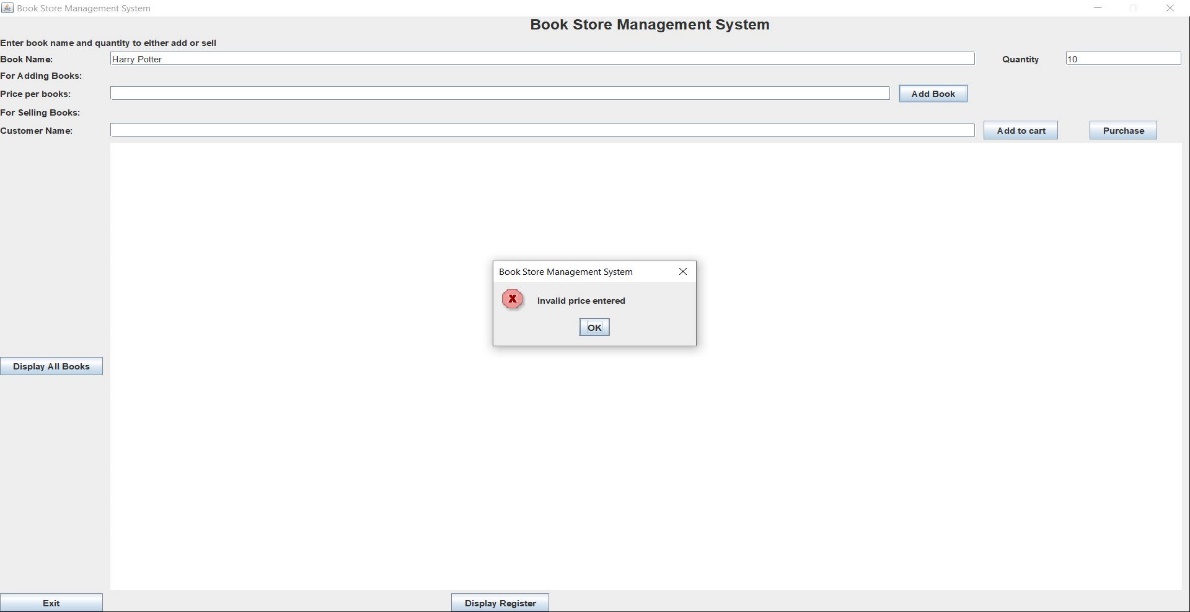


1. If there are no book available, clicking Display All Books button will display this error.

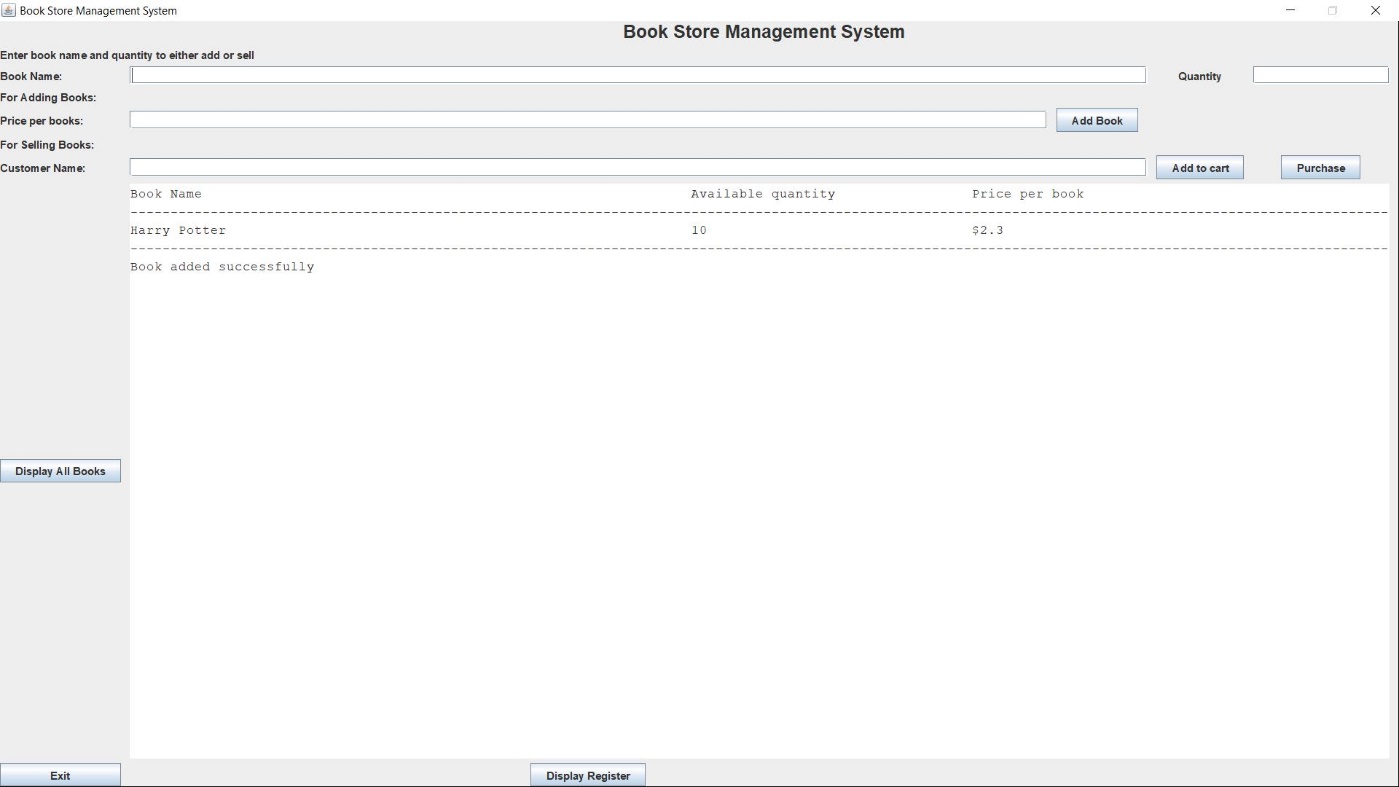


2. If there are no book sold yet, clicking Display Register button will display this error.

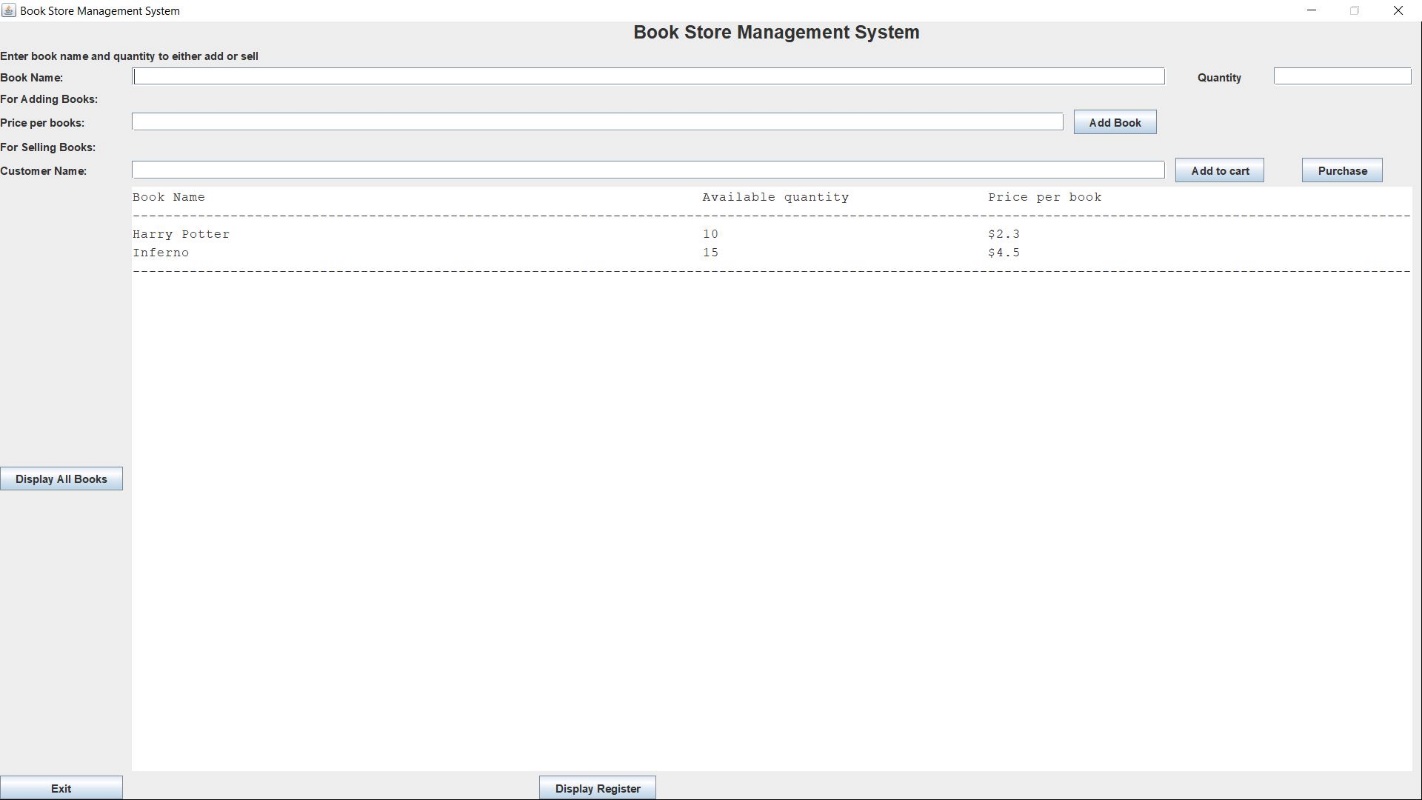
 



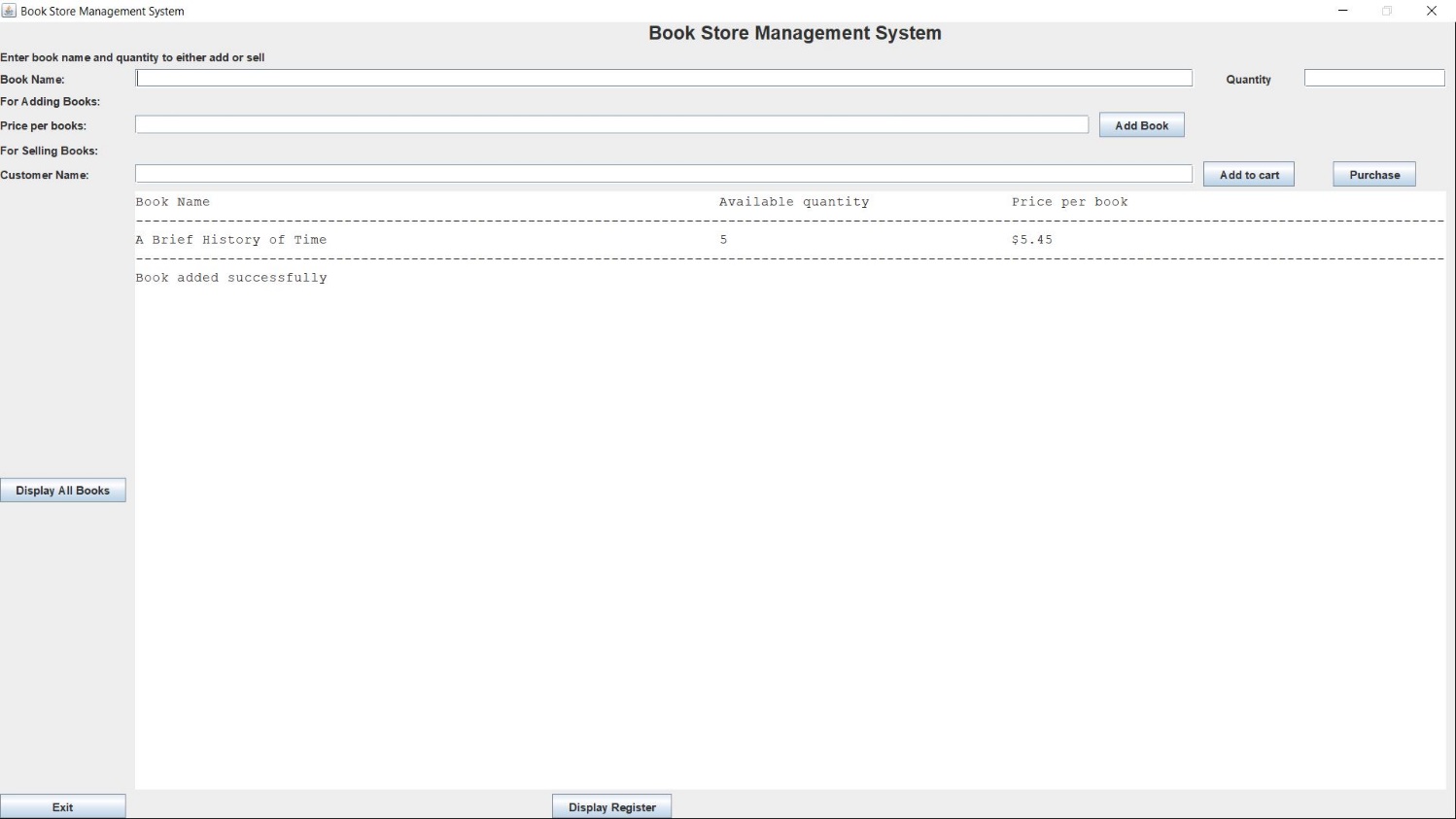
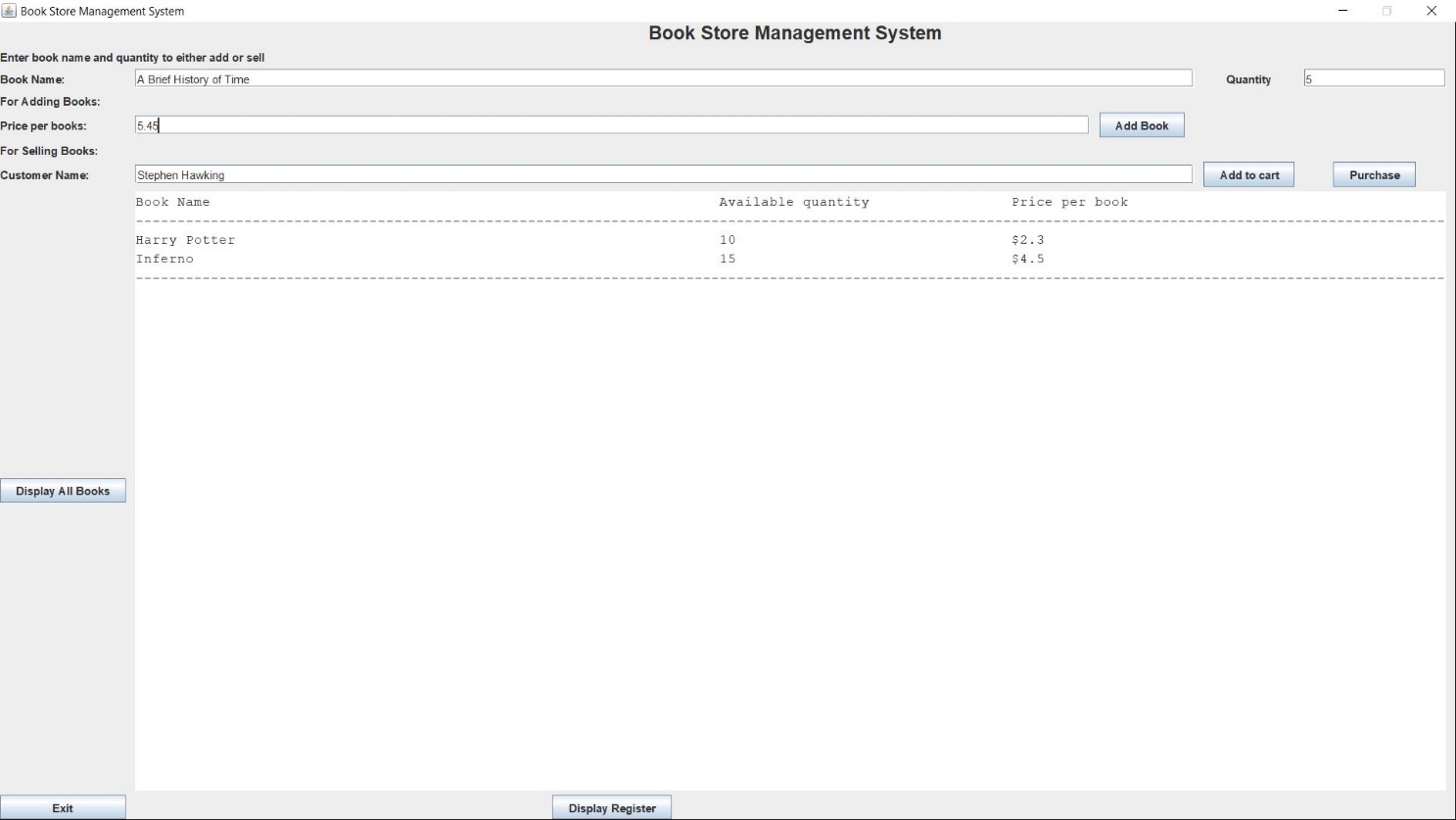
3. If no book name, invalid quantity (blank or string or float) and invalid price (blank or string) input given and Add Books button is clicked, these error dialogue will be displayed.



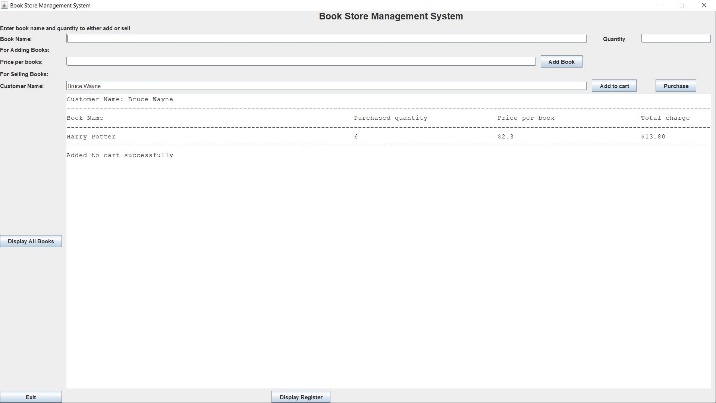
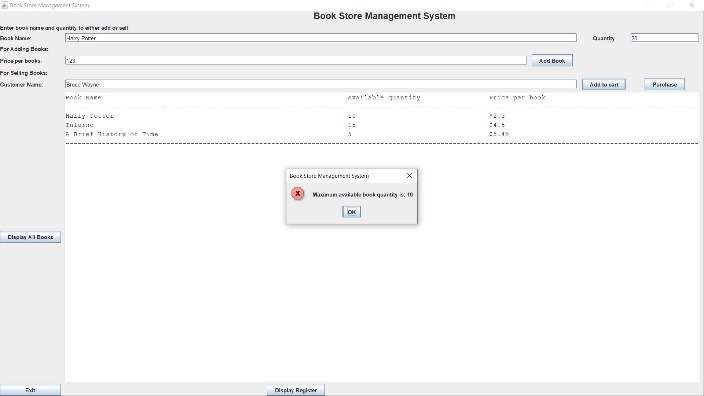
4. If inputs are correct, clicking Add Books button will display added book information in the text area.



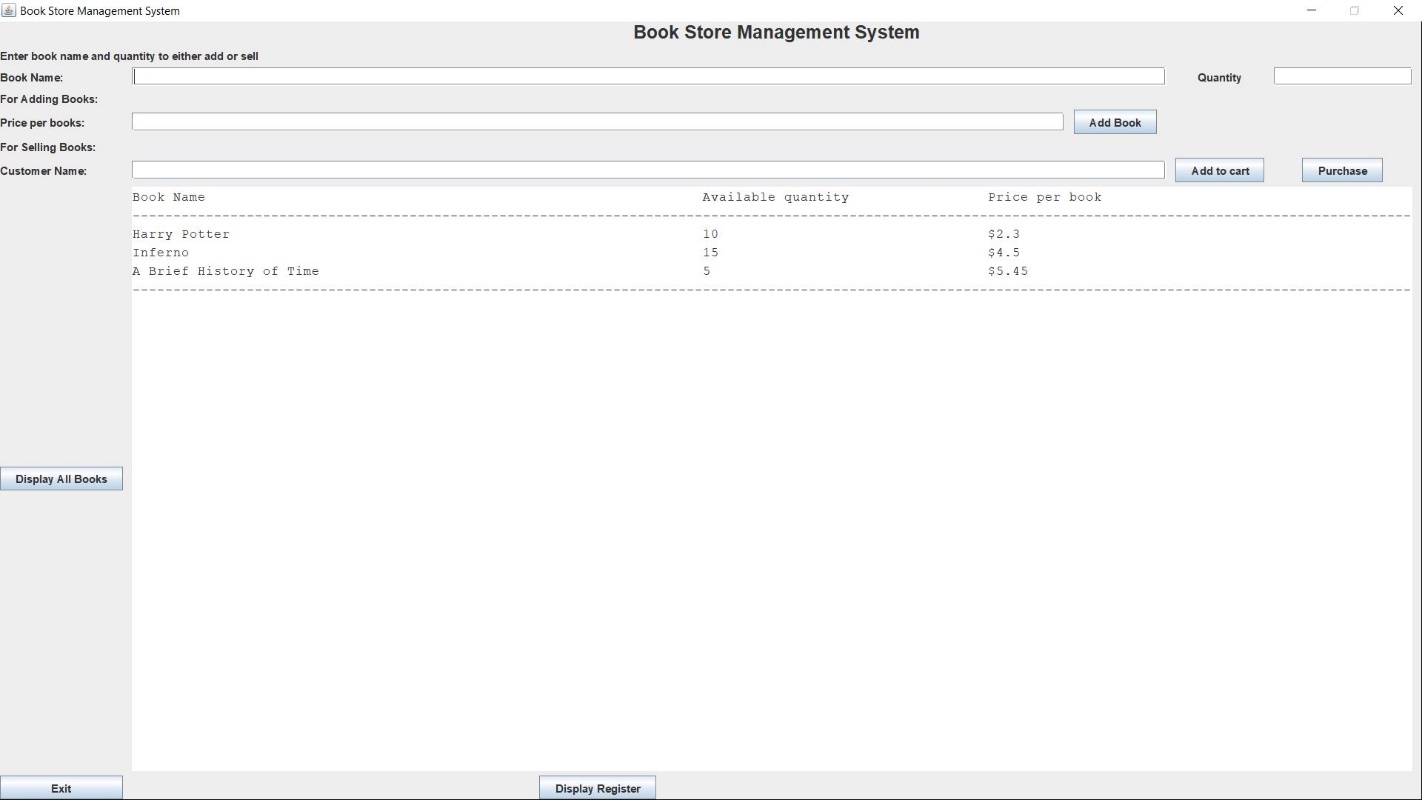
5. After adding another book to the store, Display All Books will display 2 books information in the text area.

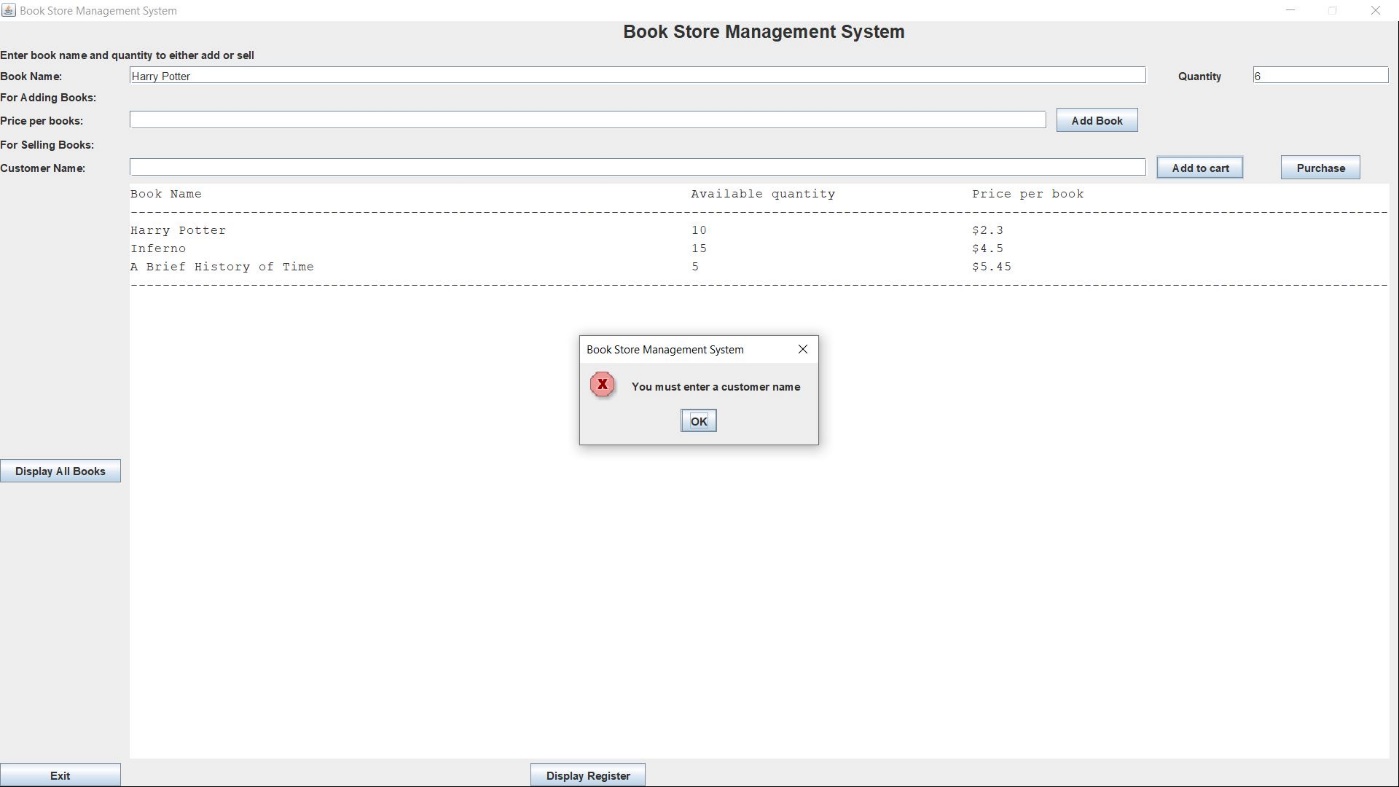


6. While adding a book in the store, if there are any input in customer name field, it will be ignored and only book name, quantity and price correct input will be taken. Similarly, while adding a book in the cart, if there are any input in price field, it will be ignored and only book name, quantity and customer name correct input will be taken.

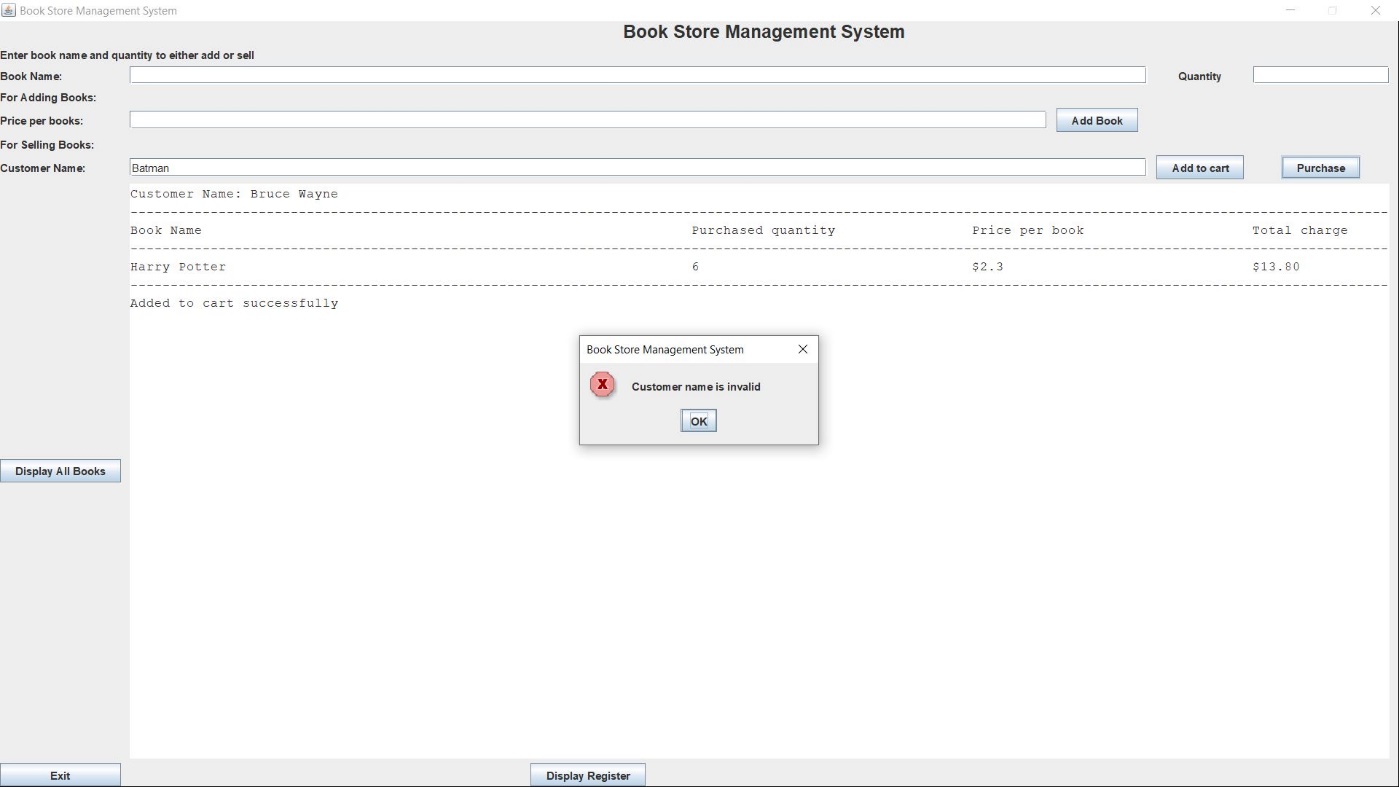


7. There are only 10 Harry Potter books available in the store. Attempting to add 20 books results in error. Adding 6 is allowed and added book information is displayed with customer name in the text area.

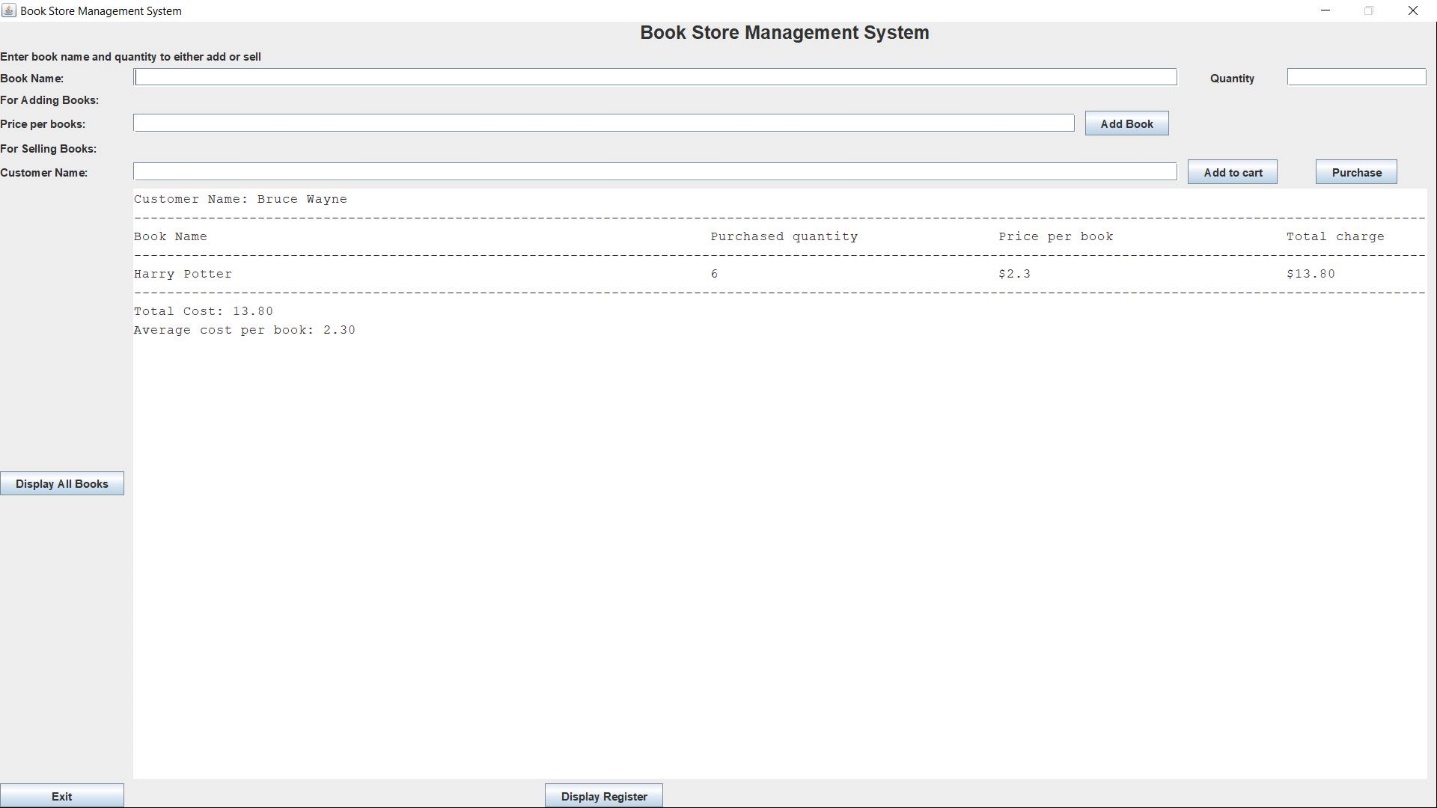


8. Harry Potter book is added to the cart but before clicking purchase button, clicking Display All button emptied the cart. Therefore, available Harry Potter book quantity is not decreased.

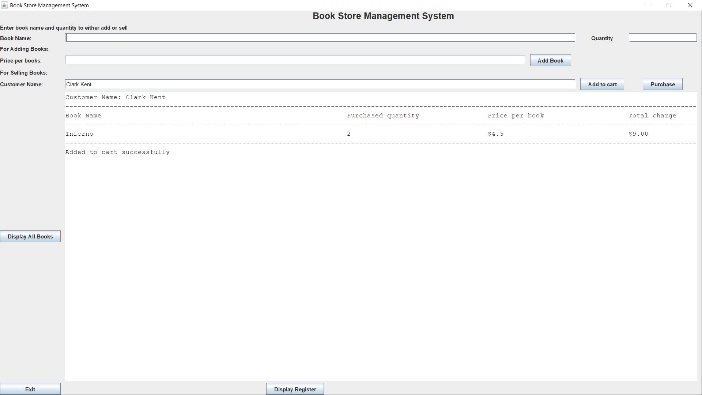
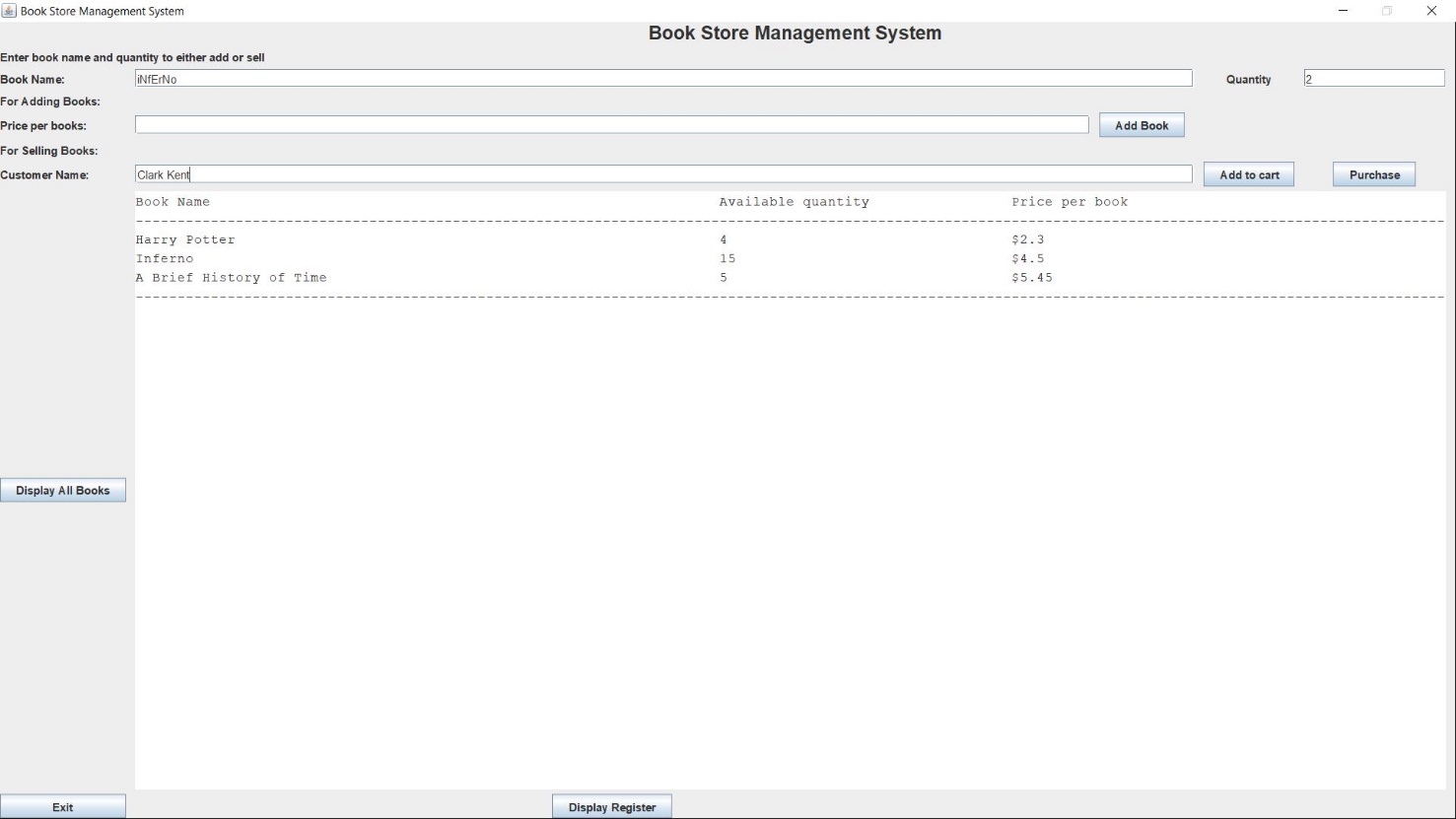
9. Clicking Add to cart without entering customer name result in error.



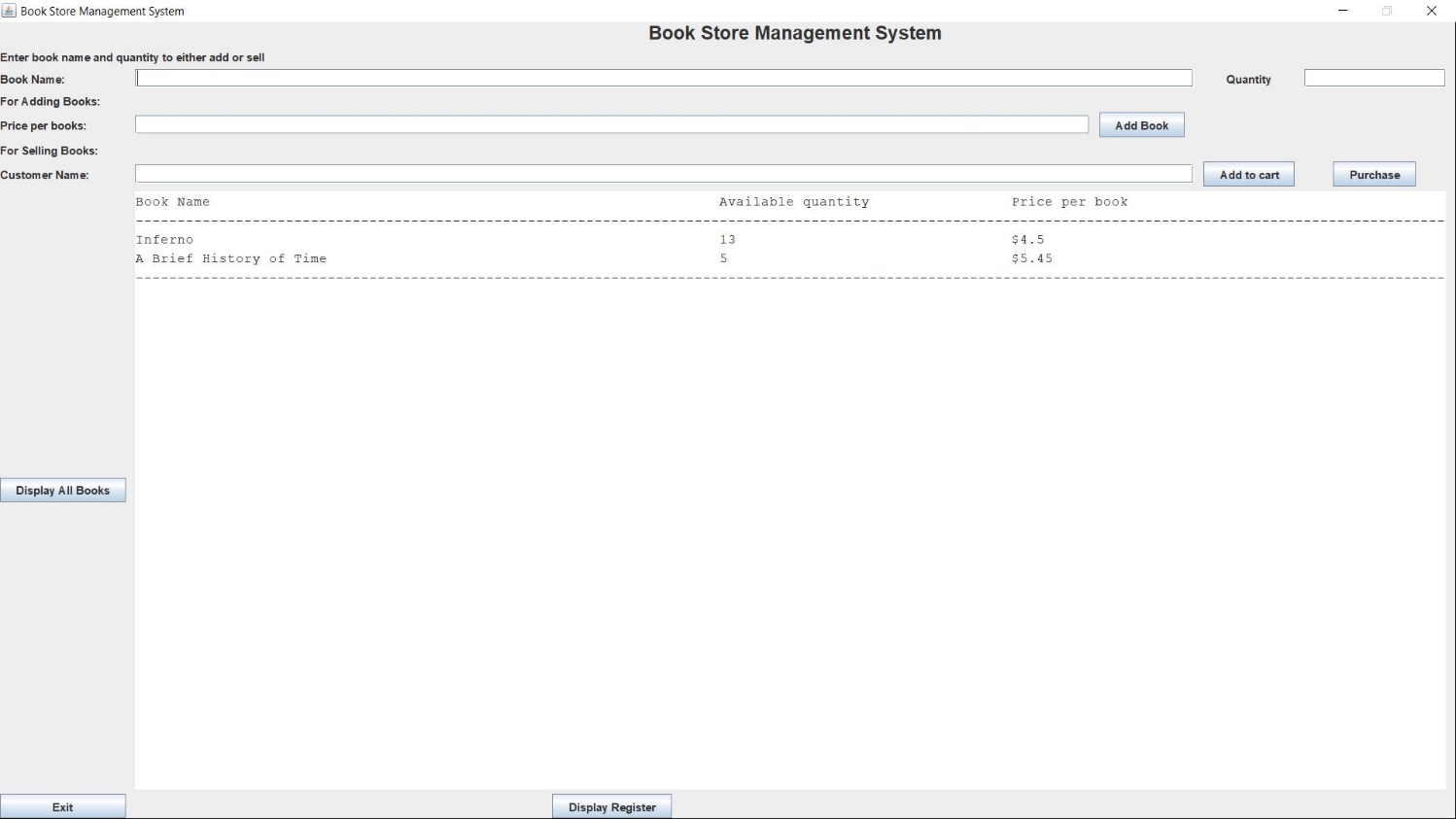
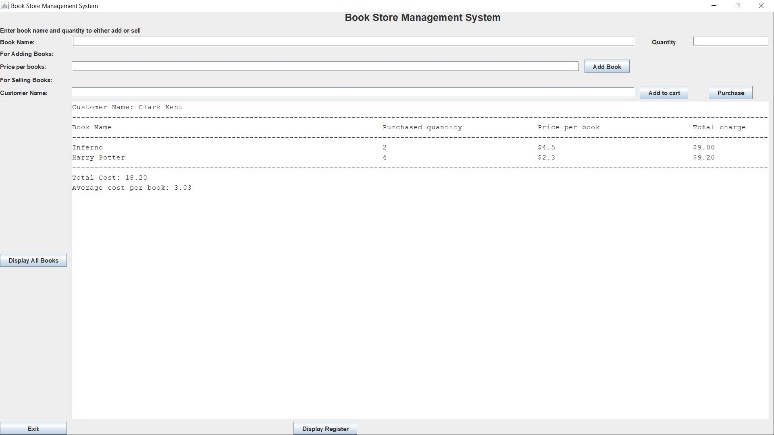
10. 6 Harry Potter books are added to the cart by the customer “Bruce Wayne”. Before clicking purchase button customer name is edited to “Batman” and it causes the error.



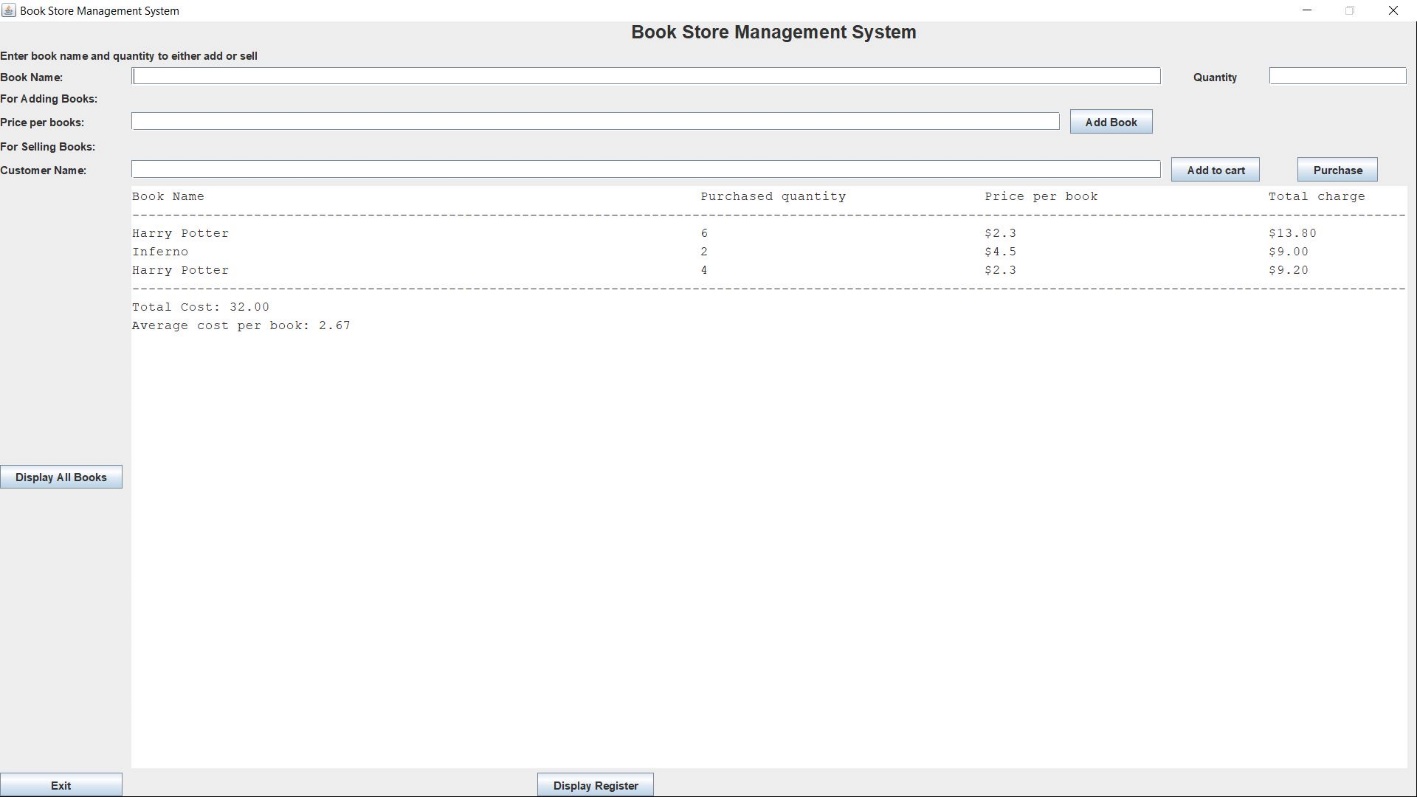
11. After successfully purchasing books, customer name, purchased book information and total cost of all books and average cost per books are displayed in the text area.



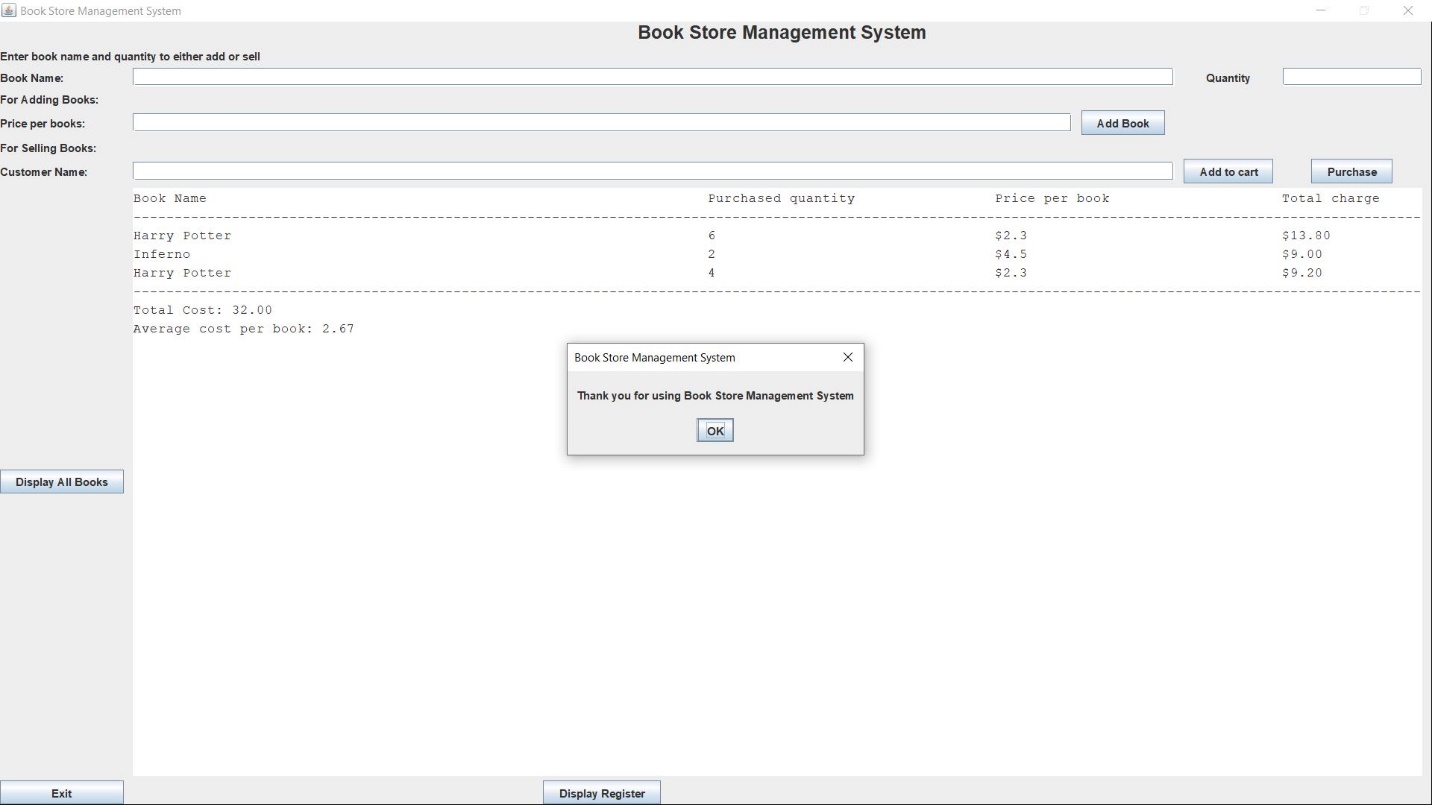
12. As 6 Harry Potter books are purchased, not only 4 Harry potter books are available shown in text area after clicking Display All Books. Customer “Clark Kent” wish to by 2 Inferno books but he searched the books in case insensitive format. However, he is still able to add the books to the cart.



13. Clark Kent also brought the last 4 available Harry Potter books. After that there are no harry Potter books available in the store.



14. Clicking Display Register button displays all the books information purchased by customer one by one. It also displays total cost of all purchased books and average cost of purchased books in the text area.



15. Exiting the program will show a thank you dialogue.

**Marking Scheme**

1. Variables: Variables have meaningful names and use camel notation. Variables are the correct type and constants are used. Data structure like ArrayList is properly declared and used.

Marks: 2

1. Code is indented and aligned correctly and is easy to read (use of vertical whitespace). Each source file has a header comment which includes name, student ID, date, file name and purpose of the class. Code is fully commented including all variables and methods. No repeated code, printing and error messages

Marks: 3

1. Instance variables are correct and private in Customer and Book class. Default and parameterised constructors are correct. Get and set methods are correct.

Marks: 2

1. All input is read correctly. For any error, input fields are cleared and focus set on right field. No error is shown because of taking faulty input.

Marks: 3

1. After handling each error, error dialogue is shown properly. Exit message is displayed.

Marks: 3

1. All total cost, average cost calculation is accurately done. All operation: Add Books, Add to cart, Purchase, Display All Books and Display Register are performed correctly by the program. All data is correctly displayed in the text area. Text area displays data in a correct format.

Marks: 7

**Appendix**

**Note:** BookStoreManagementSystem.form has the design of the GUI. Can not be added to the appendix.

**BookStoreManagementSystem.java** (include main method)

/\*

Name:

ID:

Course:

File: BookStoreManagementSystem.java

Purpose: Project -- Book Store Management System application

Date:

\*/

import javax.swing.\*;  
  
import java.awt.event.ActionListener;  
import java.awt.event.ActionEvent;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import java.util.ArrayList;  
  
  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JButton;  
import javax.swing.JTextField;  
import javax.swing.JTextArea;  
import javax.swing.JOptionPane;  
  
public class BookStoreManagementSystem extends JFrame implements ActionListener  
{  
 private ArrayList<Book> book = new ArrayList<Book>(); // ArrayList to store all available book information  
 private ArrayList<Customer> customer = new ArrayList<Customer>(); // ArrayList to store all customer name and purchased book information  
 private ArrayList<Book> bookCart = new ArrayList<Book>(); // ArrayList to store book added to the cart by customer  
 private String customerName; // Store customer name. Needed for verification during purchase after book added to cart  
  
 private JTextField bookNameField;  
 private JTextField priceField;  
 private JButton addBookButton;  
 private JTextField customerNameField;  
 private JButton addToCartButton;  
 private JButton purchaseButton;  
 private JTextArea displayTextArea;  
 private JTextField quantityField;  
 private JLabel headingLabel1;  
 private JLabel bookNameLabel;  
 private JLabel hiddenLabel;  
 private JButton displayAllBooksButton;  
 private JButton exitButton;  
 private JLabel headingLabel2;  
 private JLabel priceLabel;  
 private JLabel headingLabel3;  
 private JLabel customerNameLabel;  
 private JPanel rootPanel;  
 private JLabel quantityLabel;  
 private JButton displayRegisterButton;  
  
 public BookStoreManagementSystem () {  
 setDefaultCloseOperation(JFrame.*DO\_NOTHING\_ON\_CLOSE*); // allow the code to close the program  
 setBounds(0, 0, 1000, 1500); // Define position and size of app  
 setTitle("Book Store Management System"); // Set the title of the app  
 setExtendedState(JFrame.*MAXIMIZED\_BOTH*); // Make the application maximized screened  
 setResizable(false); // Make the application not resizable  
  
 add(rootPanel); // add panel to JFrame  
  
 addBookButton.addActionListener(this); // add the action listener to the buttons  
 addToCartButton.addActionListener(this);  
 displayAllBooksButton.addActionListener(this);  
 displayRegisterButton.addActionListener(this);  
 purchaseButton.addActionListener(this);  
 exitButton.addActionListener(this);  
  
 // when the user pushes the system close (X top right corner)  
 addWindowListener( // override window closing method  
 new WindowAdapter() {  
 public void windowClosing(WindowEvent e) {  
 exit(); // Attempt to exit application  
 }  
 }  
 );  
  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e)  
 { // process the clicks on all of the buttons  
 String command = e.getActionCommand();  
 if (command.compareTo("Add to cart") == 0)  
 addToCart();  
 else if (command.compareTo("Add Book") == 0)  
 addBooks();  
 else if (command.compareTo("Purchase") == 0)  
 purchase();  
 else if (command.compareTo("Display All Books") == 0)  
 displayAll();  
 else if (command.compareTo("Display Register") == 0)  
 displayRegister();  
 else if (command.compareTo("Exit") == 0)  
 exit();  
 }  
  
 // This method takes book name, quantity and price as input and store it in an object of Book class  
 // Display added book to the store in TextArea  
 private void addBooks()  
 {  
  
 // Check if book name is entered or not. If not then end the execution of method  
 if(!checkBookName())  
 {  
 clearCart(); // Book cart ArrayList is cleared  
 return;  
 }  
  
 // Else book name is retrieved from name field  
 String bookName = bookNameField.getText(); // Store Book name  
  
 // Check if book quantity is valid or not. If not then end the execution of method  
 if(!checkBookQuantity())  
 {  
 clearCart(); // Book cart is cleared  
 return;  
 }  
 clearCart(); // Book cart is cleared  
  
 int quantity = Integer.*parseInt*(quantityField.getText()); // Store available book quantity  
  
 // If invalid price or no price is entered, exception is caught and error dialogue with appropriate message is displayed  
 // Focus is set in the price field  
 // Else price per book is retrieved from price field and converted to double  
 double price = 0.0; // Store price per book  
 try  
 {  
 price = Double.*parseDouble*(priceField.getText());  
 }  
 catch(Exception e)  
 {  
 String errorMessage = "Invalid price entered";  
 showError(errorMessage);  
 priceField.setText("");  
 customerNameField.setText("");  
 priceField.requestFocus();  
 clearCart(); // Book cart is cleared  
 return;  
 }  
  
 // Store book information in book class object  
 Book b = new Book(bookName, quantity, price);  
  
 // Store book object to book ArrayList  
 book.add(b);  
 displayAvailableBookHeading(); // Display book information header of Text Area  
  
 // Get the book index from book ArrayList  
 int index = book.indexOf(b);  
  
 // Display book information of given index and success message in Text Area  
 displayAvailableBookData(index);  
 appendLine();  
 displayTextArea.append("Book added successfully\n");  
  
 // All the input field of GUI is restored and focus is set in the book name field  
 // Book cart is cleared  
 reset();  
 clearCart();  
 }  
  
 // This method takes book name, purchase quantity and customer name, and add this to book cart ArrayList and display book information  
 private void addToCart()  
 {  
 // If customer name is empty error dialogue is displayed with appropriate message  
 // Price field and customer field is cleared  
 // Focus is set on customer name field  
 if (customerNameField.getText().compareTo("") == 0)  
 {  
 String errorMessage = "You must enter a customer name";  
 showError(errorMessage);  
 customerNameField.setText("");  
 priceField.setText("");  
 customerNameField.requestFocus();  
 return;  
 }  
  
 // If book cart is not empty but customer name is changed it will clear the book cart  
 if(!bookCart.isEmpty() && !customerNameField.getText().equals(customerName))  
 {  
 bookCart.clear();  
 }  
 // Else Customer name is retrieved from name field  
 customerName = customerNameField.getText();  
  
 // Check if book name is entered or not. If not then end the execution of method  
 if(!checkBookName())  
 {  
 return;  
 }  
  
 // Else book name is retrieved from name field  
 String bookName = bookNameField.getText(); // Store Book name  
  
 // Check if book quantity is valid or not. If not then end the execution of method  
 if(!checkBookQuantity())  
 {  
 return;  
 }  
  
 int quantity = Integer.*parseInt*(quantityField.getText()); // Store available book quantity  
  
 // If quantity is zero, error dialogue is displayed with appropriate message  
 // Price field and quantity field is cleared and focus is set on quantity field  
 if(quantity == 0)  
 {  
 String errorMessage = "Invalid quantity entered";  
 showError(errorMessage);  
 priceField.setText("");  
 quantityField.setText("");  
 quantityField.requestFocus();  
 return;  
 }  
  
 // Check if the book is available in the book ArrayList  
 boolean bookFound = false; // If book is found, make this true  
  
 for(int i = 0; i < book.size(); i++)  
 {  
 // Check if entered book is available in book ArrayList  
 if(book.get(i).getBookName().equalsIgnoreCase(bookName))  
 {  
 bookFound = true;  
 // Check asking book quantity is smaller or equal to available book quantity  
 if(book.get(i).getQuantity() >= quantity)  
 {  
 bookCart.add(new Book(book.get(i).getBookName(), quantity, book.get(i).getPrice())); // Add book object to bookCart ArrayList  
 }  
 // Else error message is shown with appropriate dialogue  
 // Price field and quantity field is cleared and focus is set on quantity field  
 else  
 {  
 String errorMessage = "Maximum available book quantity is: "+book.get(i).getQuantity();  
 showError(errorMessage);  
 priceField.setText("");  
 quantityField.setText("");  
 quantityField.requestFocus();  
 return;  
 }  
 }  
 }  
 // If book is unavailable error message is shown with appropriate dialogue  
 // Book name field, price field and quantity field is cleared and focus is set on book name field  
 if(!bookFound)  
 {  
 String errorMessage = bookName + " is not available";  
 showError(errorMessage);  
 reset();  
 customerNameField.setText(customerName);  
 return;  
 }  
  
 // Display added to cart book information and success message in the TextArea  
 displayCustomerHeading(customerName);  
 displayCartBookData(bookCart.size()-1);  
 appendLine();  
 displayTextArea.append("Added to cart successfully\n");  
 reset(); // All input field is restored  
 customerNameField.setText(customerName);  
  
 }  
  
 // This method purchase all the books added to the book cart ArrayList by adding customer name and book information in customer ArrayList  
 // Displays all book purchased by customer in TextArea  
 private void purchase()  
 {  
 // If bookCart is empty, error dialogue is displayed with appropriate message  
 if(bookCart.isEmpty())  
 {  
 String errorMessage = "Cart is empty";  
 showError(errorMessage);  
 reset(); // All Gui input field is restored and focus is set on book name  
 return;  
 }  
  
 // If customer name does not matches with previous one, error dialogue is displayed with appropriate message  
 if(!customerNameField.getText().equals(customerName))  
 {  
 String errorMessage = "Customer name is invalid";  
 showError(errorMessage);  
 reset(); // All GUI input field is restored and focus is set on book name  
 return;  
 }  
  
 for(int i = 0; i < bookCart.size(); i++)  
 {  
 for(int j = 0; j < book.size(); j++)  
 {  
 // Check if entered book is available in book ArrayList  
 // Book name search is case insensitive  
 if(book.get(j).getBookName().equalsIgnoreCase(bookCart.get(i).getBookName()))  
 {  
 // If available book quantity is more then entered quantity, available quantity is subtracted from entered quantity  
 if(book.get(j).getQuantity() > bookCart.get(i).getQuantity())  
 {  
 book.get(j).setQuantity((book.get(j).getQuantity() - bookCart.get(i).getQuantity()));  
 }  
  
 // If available book quantity is equal to entered quantity, book is removed from book ArrayList  
 else if(book.get(j).getQuantity() == bookCart.get(i).getQuantity())  
 {  
 book.remove(j);  
 }  
 }  
 }  
 }  
  
 // Display customer name and purchased book heading and information  
 displayCustomerHeading(customerName);  
 displayCustomerRegister();  
  
 // Customer name and book information is added to the customer ArrayList  
 for(int i = 0; i < bookCart.size(); i++)  
 {  
 Customer temp = new Customer(customerName, bookCart.get(i).getBookName(), bookCart.get(i).getQuantity(), bookCart.get(i).getPrice());  
 customer.add(temp);  
 }  
 // Clear cart and reset all input field and focus is set on book name  
 clearCart();  
 reset();  
 }  
  
 // This method display all the available boon information in the TextArea  
 private void displayAll()  
 {  
 // If no book is added, error message is displayed with appropriate dialogue  
 if(book.size()==0)  
 {  
 String errorMessage = "No book available";  
 showError(errorMessage);  
 reset(); // All GUI input field is restored and focus is set on book name  
 return;  
 }  
  
 // Display available book header and all book information in the TextArea  
 displayAvailableBookHeading();  
 for(int i = 0; i < book.size(); i++)  
 {  
 displayAvailableBookData(i);  
 }  
 appendLine();  
  
 // Clear cart and reset all input field and focus is set on book name  
 reset();  
 clearCart();  
 }  
  
 private void displayRegister()  
 {  
 // Clear cart, TextArea and reset all input field and focus is set on book name  
 reset();  
 clearCart();  
 clearTextArea();  
  
 // If no book it entered, error dialogue is displayed with appropriate message  
 if(customer.isEmpty())  
 {  
 String errorMessage = "Register is empty";  
 showError(errorMessage);  
 reset(); // All Gui input field is restored and focus is set on book name  
 return;  
 }  
  
 displayPurchasedBookHeading(); // Display purchased book heading in the TextArea  
  
 // Add all book information brought by customers in book cart ArrayList  
 for(int i = 0; i < customer.size(); i++)  
 {  
 Book temp = new Book(customer.get(i).getBookName(), customer.get(i).getQuantity(), customer.get(i).getPrice());  
 bookCart.add(temp);  
 }  
  
 // Display all the books purchased by customers in the TextArea  
 displayCustomerRegister();  
 }  
  
 // If book name is empty error dialogue is displayed with appropriate message and return false.  
 // Else return true.  
 private boolean checkBookName()  
 {  
 if(bookNameField.getText().compareTo("") == 0)  
 {  
 String errorMessage = "You must enter a book name";  
 showError(errorMessage);  
 reset(); // All the input field of GUI is restored and focus is set in the book name field  
 return false;  
 }  
 return true;  
 }  
  
 // If invalid quantity or no quantity is entered, exception is caught and error dialogue with appropriate message is displayed and return false.  
 // Focus is set in the quantity field  
 // Price field, quantity field and customer field is cleared  
 // Else quantity is retrieved from quantity field and converted to integer and return true  
 private boolean checkBookQuantity()  
 {  
 int q = 0;  
 try  
 {  
 q = Integer.*parseInt*(quantityField.getText()); // Store available book quantity  
 return true;  
 }  
 catch(Exception e)  
 {  
 String errorMessage = "Invalid quantity entered";  
 showError(errorMessage);  
 quantityField.setText("");  
 priceField.setText("");  
 customerNameField.setText("");  
 quantityField.requestFocus();  
 return false;  
 }  
  
 }  
  
 // This method display header for showing available books in appropriate format in the Text Area  
 // Append a separator line  
 private void displayAvailableBookHeading()  
 {  
 displayTextArea.setText(String.*format*("%-70s%-35s%-35s\n", "Book Name", "Available quantity", "Price per book"));  
 appendLine();  
 }  
  
 // This method display header for showing added to cart / purchased books in appropriate format in the Text Area  
 // Append a separator line  
 private void displayPurchasedBookHeading()  
 {  
 displayTextArea.append(String.*format*("%-70s%-35s%-35s%s\n", "Book Name", "Purchased quantity", "Price per book", "Total charge"));  
 appendLine();  
 }  
  
 // This method display customer name in the Text Area  
 // Append a separator line  
 private void displayCustomerHeading(String name)  
 {  
 displayTextArea.setText("Customer Name: " + name + "\n");  
 appendLine();  
 displayPurchasedBookHeading();  
 }  
  
 // Gets the book name, available quantity and price from the given book index and display in the text area with appropriate format  
 private void displayAvailableBookData(int index)  
 {  
 displayTextArea.append(String.*format*("%-70s%-35s%-35s\n", book.get(index).getBookName(), book.get(index).getQuantity(), "$"+book.get(index).getPrice()));  
 }  
  
 // Gets the book name, available quantity and price from the given bookCart index and display in the text area with appropriate format  
 private void displayCartBookData(int index)  
 {  
 displayTextArea.append(String.*format*("%-70s%-35s%-35s%s\n", bookCart.get(index).getBookName(), bookCart.get(index).getQuantity(), "$" + bookCart.get(index).getPrice(), "$" + String.*format*("%.2f",(bookCart.get(index).getPrice() \* bookCart.get(index).getQuantity()))));  
 }  
  
 // This method display purchased book information  
 private void displayCustomerRegister()  
 {  
 double total\_price = 0.0; //Total cost of all books  
 int total\_quantity = 0; //Total number of purchased books  
  
 // Display all book purchased by customer  
 for(int i = 0; i < bookCart.size(); i++)  
 {  
 displayCartBookData(i);  
 total\_price += (bookCart.get(i).getPrice() \* bookCart.get(i).getQuantity());  
 total\_quantity += bookCart.get(i).getQuantity();  
 }  
  
 appendLine();  
  
 // Display Total Cost and Average cost per book on purchased books  
 displayTextArea.append("Total Cost: "+String.*format*("%.2f",total\_price) + "\n");  
 displayTextArea.append("Average cost per book: "+String.*format*("%.2f",(total\_price / total\_quantity)) + "\n");  
 }  
  
 // This method separate line of information  
 private void appendLine()  
 {  
 displayTextArea.append("-------------------------------------------------------------------------------------------------------------------------------------------------------------\n");  
 }  
  
 // All the input field of GUI is restored and focus is set in the Book name field  
 private void reset()  
 {  
 bookNameField.setText("");  
 quantityField.setText("");  
 priceField.setText("");  
 customerNameField.setText("");  
 bookNameField.requestFocus();  
 }  
  
 // Clear bookCart ArrayList and customer name  
 private void clearCart()  
 {  
 bookCart.clear();  
 customerName = "";  
 }  
  
 // Clear Text Area  
 private void clearTextArea()  
 {  
 displayTextArea.setText("");  
 }  
  
 // Take message in the parameter and show it in the error dialogue  
 private void showError(String errorMessage)  
 {  
 JOptionPane.*showMessageDialog*(new JFrame(), errorMessage, "Book Store Management System",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 // Exit the application by showing a thank you message in the dialogue box  
 private void exit()  
 {  
  
 // *TODO -- display exit message here* JOptionPane.*showMessageDialog*(new JFrame(), "Thank you for using Book Store Management System", "Book Store Management System",  
 JOptionPane.*PLAIN\_MESSAGE*);  
 System.*exit*(0);  
 } // exit  
  
 public static void main(String args [])  
 {  
 BookStoreManagementSystem f = new BookStoreManagementSystem(); // Create instance of class  
 f.setVisible(true); // Make the application visible  
 }  
}

**Customer.java**

/\*

Name:

ID:

Course:

File: Customer.java

Purpose: Customer class to store customer name and purchased book name, quantity and price per book

Date:

\*/

public class Customer

{

// Variables to store customer name and book information

private String customerName;

private String bookName;

private int quantity;

private double price;

// Constructor to set customer namem book information during object creation

public Customer(String customerName, String bookName, int quantity, double price) {

this.customerName = customerName;

this.bookName = bookName;

this.quantity = quantity;

this.price = price;

}

// Getters and setters to get and set customer name and book name, purchased quantity and price per book in an ArrayList from outside of customer class

public String getCustomerName() {

return customerName;

}

public void setCustomerName(String customerName)

{

this.customerName = customerName;

}

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**Book.java**

/\*

Name:

ID:

Course:

File: Book.java

Purpose: Book class to store book name, available quantity and price per book as an object

Date:

\*/

public class Book

{

// Variables to store book information

private String bookName;

private int quantity;

private double price;

// Constructor to set book information during object creation

public Book(String bookName, int quantity, double price)

{

this.bookName = bookName;

this.quantity = quantity;

this.price = price;

}

// Getters and setters to get and set book information like name, available quantity and price per book from outside of book class

public String getBookName() {

return bookName;

}

public void setBookName(String bookName) {

this.bookName = bookName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}