Practical Assignment

Voningo Flora Makhubele Pecego

ST10483982

School of Computer Science, Varsity College

PROG6112: Programming 1B

Lecturer: Amakan Elisha Agoni

Group 3

Date: 10 September 2025

Table of Contents

GitHub Repository

Link to Repository

https://github.com/VCCT-PROG6112-2025-G3/Repo_PracticalAssignment_ST10483982_PROG6112_GR03.git

Section A: TV Series Management Application

Please note: for clarity and readability, I have changed the title blocks and comments from black to grey text.

Complete Code for Section A – Question 1

Main.java

```
/*
================================================================================================
============================================================*/
/* TV Series Management Application
/*
================================================================================================
============================================================*/
/* Name: Voningo Flora Makhubele Pecego
/* Student Number: ST10483982
/* Assignment: Practical Assignment - Section A
/* Lecturer: Amakan Elisha Agoni (BCA1 GRO3)
/* Date: 04/09/2025
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* MAIN METHOD (Main.java)
/*
/* Description: Main class that serves as the entry point for the TV Series Management Application.
/*              It launches the menu system by calling methods from the Menu class, which controls the
overall flow of the program.
/* ------------------------------------------------------------------------------------------------------------------------
--------*/

package tvseriesapplication;

public class Main {

   public static void main(String[] args) {
      // Start running application
      new Menu().startRunning();
   }

}

// ------------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                          //
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

Menu.java

```
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* MENU CLASS (Menu.java)                                                                                              /
```

```
/*                                                                                                    /
/* Description: Menu class that provides methods for navigating the TV Series Management
Application.                          /
/*          It enables the welcome screen, main menu, and exit confirmation to be displayed, and
contains the startRunning()       /
/*          method which delegates user actions (capture, search, update, delete, recieve report, exit) to
the Series class.       /
/* ----------------------------------------------------------------------------------------------------------------------
--------*/

package tvseriesapplication;

import javax.swing.*;

public class Menu {

   private final Series seriesApp = new Series();
   public void startRunning(){
      // Welcome screen ("1" to continue; any other key exits)
      if (!showWelcomeMenu()){
         if (confirmExit()){
            // Calls exitSeriesApplication() method from Series class to display goodbye message if the
user confirms that they want to exit
            seriesApp.exitSeriesApplication();
            return;                                  // User exits app completely
         }
      }

      boolean isRunning = true;                              // i.e. if the user entered "1"
      while (isRunning){
         int userChoice = showMainMenu();

         switch (userChoice){
            case 0: seriesApp.captureSeries(); break;
            case 1: seriesApp.searchSeries(); break;
            case 2: seriesApp.updateSeries(); break;
            case 3: seriesApp.deleteSeries(); break;
            case 4: seriesApp.seriesReport(); break;
            case 5: // User pressed exit button
            case JOptionPane.CLOSED_OPTION:
               if (confirmExit()){
                  seriesApp.exitSeriesApplication();
                  isRunning = false;
               }
               break;
            default:
               // defualt for any unexpeced value
               if (confirmExit()){
                  seriesApp.exitSeriesApplication();
                  isRunning = false;
               }
               break;
         }
      }
   }
```

```
    // Method displays welcome screen. User must enter (1) to continue (true); any other key returns
false
    private boolean showWelcomeMenu(){
        String userInput = JOptionPane.showInputDialog(
                null,
                "Enter (1) to launch menu or any other key to exit",
                "LATEST SERIES - 2025",
                JOptionPane.QUESTION_MESSAGE
        );
        return userInput != null && userInput.trim().equals("1");
    }

    // Method allows user to choose what action they would like to perform (capture, search, update,
delete, receive report, exit)
    private int showMainMenu(){
        // User of numbered buttons eliminates the need to validate user input (i.e do not have to ensure
that the user enters an integer between 1 and 6)
        Object[] menuOptions = {"1", "2", "3", "4", "5", "6"};

        String optionsText =
            "Please select one of the following menu items:\n\n" +
                "(1) Capture a new series\n" +
                "(2) Search for a series\n" +
                "(3) Update series\n" +
                "(4) Delete a series\n" +
                "(5) Print series report - 2025\n" +
                "(6) Exit Application";

        return JOptionPane.showOptionDialog(
                null,
                optionsText,
                "Manage TV Series",
                JOptionPane.DEFAULT_OPTION,
                JOptionPane.PLAIN_MESSAGE,
                null,
                menuOptions,
                menuOptions[0]
        );
    }

    // Method confirms if the user wants to exit
    private boolean confirmExit(){
        int choice = JOptionPane.showConfirmDialog(
            null,
            "Are you sure you want to exit?",
            "Exit Confirmation",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE
        );
        return choice == JOptionPane.YES_OPTION;
    }

}

// ------------------------------------------------------------------------------------------------------------------
--------- //
// END OF FILE                                                                                                      //
```

```
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

SeriesModel.java

```
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* SERIES MODEL CLASS (SeriesModel.java)                                                                    /
/*                                                                                                          /
/* Description: SeriesModel class that acts as a data model representing a single TV series. It
encapsulates attributes (ID, Name, Age    /
/*           Restriction, Number of Episodes) and provides getter and setter methods, along with a
toString() method for displaying.   /                          /
/* ------------------------------------------------------------------------------------------------------------------------
--------*/

package tvseriesapplication;


public class SeriesModel {
   private String seriesId;
   private String seriesName;
   private int seriesAge;
   private int seriesNumberOfEpisodes;

/* ------------------------------------------------------------CONSTRUCTOR-----------------------------------------
-------------------*/
   public SeriesModel(String seriesId, String seriesName, int seriesAge, int seriesNumberOfEpisodes){
      this.seriesId = seriesId;
      this.seriesName = seriesName;
      this.seriesAge = seriesAge;
      this.seriesNumberOfEpisodes = seriesNumberOfEpisodes;
   }

/* ------------------------------------------------------------GETTER METHODS--------------------------------------
--------------------*/
   public String getSeriesId(){
      return seriesId;
   }

   public String getSeriesName(){
      return seriesName;
   }

   public int getSeriesAge(){
      return seriesAge;
   }

   public int getSeriesNumberOfEpisodes(){
      return seriesNumberOfEpisodes;
   }

/* ------------------------------------------------------------SETTER METHODS--------------------------------------
--------------------*/
   public void setSeriesId(String seriesId){
```

```
    this.seriesId = seriesId;
  }

  public void setSeriesName(String seriesName){
    this.seriesName = seriesName;
  }

  public void setSeriesAge(int seriesAge){
    this.seriesAge = seriesAge;
  }

  public void setSeriesNumberOfEpisodes(int seriesNumberOfEpisodes){
    this.seriesNumberOfEpisodes = seriesNumberOfEpisodes;
  }

/* ------------------------------------------------------------toString() METHOD-----------------------------------------
------------------*/
  @Override
  public String toString(){
    return "SERIES ID: " + seriesId +
        "\nSERIES NAME: " + seriesName +
        "\nSERIES AGE RESTRICTION: " + seriesAge +
        "\nSERIES NUMBER OF EPISODES: " + seriesNumberOfEpisodes;
  }

}

// ------------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                          //
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

Series.java

```
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* SERIES CLASS (Series.java)                                                                                           /
/*                                                                                                                      /
/* Description: Series class that contains methods for managing TV series, including capture, search,
update, delete, report,          /
/*          and exit. It also includes helper and validation methods (e.g., age restriction, positive integer
checks)             /
/* ------------------------------------------------------------------------------------------------------------------------
--------*/

package tvseriesapplication;

import javax.swing.*;
import java.util.ArrayList;
import java.util.List;

public class Series {
  // Declaring in-memory store (i.e. list for holding all SeriesModel objects (each individual series))
  private final List<SeriesModel> seriesList = new ArrayList<>();
```

```
/* ------------------------------------------------------MANDATORY METHODS----------------------------------------
--------------------*/
  // Method for capturing new TV series each with the following attributes: ID, name, age restriction and
episode count
  public void captureSeries(){
    String id = captureIsNotEmpty("Enter the series ID");
    if (id == null) return;                              // ID capture cancelled

    String name = captureIsNotEmpty("Enter the series name");
    if (name == null) return;                            // Name capture cancelled

    int age = captureAgeRestriction();
    if (age == -1) return;                               // Age restriction capture cancelled

    int episodes = captureEpisodeCount(name);
    if (episodes == -1) return;                          // Episode count capture cancelled

    SeriesModel newSeries = new SeriesModel(id.trim(), name.trim(), age, episodes);
    seriesList.add(newSeries);

    JOptionPane.showMessageDialog(null, "Series processed successfully!!!");
  }

  // Method searches for the series by ID and diplays the result to the user
  public void searchSeries(){
    String id = captureIsNotEmpty("Enter the series ID to search");
    if (id == null) return;                              // Search cancelled

    // Calling method that loops through array until the matching ID is found
    SeriesModel foundSeries = searchByID(id.trim());

    if (foundSeries != null){
      JOptionPane.showMessageDialog(
          null,
          "Series found:\n\n" + foundSeries.toString(),
          "Search Result",
          JOptionPane.INFORMATION_MESSAGE
      );
    } else{
      JOptionPane.showMessageDialog(
          null,
          "Series with series ID " + id + " was not found!",
          "Search Result",
          JOptionPane.WARNING_MESSAGE
      );
    }
  }

  // Method allows user to update the details of a series by ID (name, age restriction, episode count)
  public void updateSeries(){
    String id = captureIsNotEmpty("Enter the series ID to update");
    if (id == null) return;                              // Update cancelled

    SeriesModel series = searchByID(id.trim());
    if (series == null) {
      JOptionPane.showMessageDialog(
```

```
            null,
            "Series with series ID " + id + " was not found!",
            "Update Result",
            JOptionPane.WARNING_MESSAGE
        );
        return;
    }

    if (!confirmUpdate(series)){
        JOptionPane.showMessageDialog(null, "Update cancelled.");
        return;
    }

    // Updating the tv series details by calling helper methods
    updateName(series);
    updateAge(series);
    updateEpisodeCount(series);

    JOptionPane.showMessageDialog(
            null,
            "Series with ID " + id + " has been successfully updated!",
            "Update Successful",
            JOptionPane.INFORMATION_MESSAGE
    );
  }

  public void deleteSeries(){
    String id = captureIsNotEmpty("Enter the series ID to delete");
    if (id == null) return;                          // Delete cancelled

    SeriesModel series = searchByID(id.trim());

    if (series == null){
        JOptionPane.showMessageDialog(
                null,
                "Series with series ID " + id + " was not found!",
                "Delete Result",
                JOptionPane.WARNING_MESSAGE
        );
        return;
    }

    // Confirming if the user wants to delete the series and removing it from the list
    if (confirmDelete(series)){
        seriesList.remove(series);
        JOptionPane.showMessageDialog(
                null,
                "Series with ID " + id + " was deleted.",
                "Delete Successful",
                JOptionPane.INFORMATION_MESSAGE
        );
    } else {
        JOptionPane.showMessageDialog(null, "Deletion of series cancelled.", "Delete Cancelled",
JOptionPane.INFORMATION_MESSAGE);
    }
  }
```

```
    // Method for diplaying a report of all of the captured series (uses HTML formatting (ChatGPT, 2025))
    public void seriesReport(){
        // Returning an error message if the list is empty
        if (seriesList.isEmpty()){
            JOptionPane.showMessageDialog(null, "No series have been captured.", "Series Report",
JOptionPane.INFORMATION_MESSAGE);
            return;
        }

        // Making use of HTML text formatting
        StringBuilder report = new StringBuilder("<html>");
        report.append("<h2>Series Report</h2>");                    // Title of report uses <h2> to make the
report title bigger and in bold

        int seriesNum = 1;
        for (SeriesModel series : seriesList) {
            report.append("<hr>")                                   // Inserting horizontal line separator (<hr>)
                .append("<b>Series ").append(seriesNum++).append("</b><br>")  // Bold text (<b>)
                .append("<hr>")
                .append("ID: ").append(series.getSeriesId()).append("<br>")
                .append("Name: ").append(series.getSeriesName()).append("<br>")
                .append("Age Restriction: ").append(series.getSeriesAge()).append("<br>")
                .append("Episodes: ").append(series.getSeriesNumberOfEpisodes()).append("<br>");
        }

        // Total series count
        report.append("<hr>")
          .append("<b>Total Series: ").append(seriesList.size()).append("</b>")
          .append("</html>");

        JOptionPane.showMessageDialog(null, report.toString(), "Series Report",
JOptionPane.PLAIN_MESSAGE);
    }

    public void exitSeriesApplication(){
        JOptionPane.showMessageDialog(null, "Exiting the TV Series Management Application. Until next
time!");
    }

/* ------------------------------------------------------VALIDATION METHODS-----------------------------------------
-------------------*/
    // Method that validates that the value input by the user is a positive integer (i.e. non-negative)
    public int checkPositiveInt(String userInput){
        // Exception Handling
        try{
            int number = Integer.parseInt(userInput);
            // ternary operator that returns the userInput if true or -1 if false
            return (number >= 0) ? number : -1;
        } catch (NumberFormatException e) {
            return -1;                                      // Invalid number
        }
    }

    // Method that validates the age restriction input by the user
    public int checkAgeRestriction(String userInput){
        // Exeption Handling
        try{
```

```
        int age = Integer.parseInt(userInput);
        // ternary operator that returns the userInput (age) if true (valid input = age input is between 2 and 18) or -1 if false
        return (age >= 2 && age <= 18) ? age : -1;
      } catch (NumberFormatException e) {
        return -1;                                  // Invalid number
      }
    }

/* ------------------------------------------------------HELPER METHODS------------------------------------------------------ */
    SeriesModel searchByID(String id){                      // id is the ID input by the user
      // Loops until all of the items on the list have been checked/searched
      for (int index = 0; index < seriesList.size(); index++){
        // Getting the element at the index called "index"
        SeriesModel series = seriesList.get(index);
        if (series.getSeriesId().equalsIgnoreCase(id)){             // Case of the ID entered by the user is ignored (i.e. not case-sensitive)
          return series;                                // Return series (object) immediately if the matching ID is found
        }
      }
      return null;                                      // No matching ID is found
    }

    private String captureIsNotEmpty(String text){
      // While loop to keep prompting the user until the user cancels or enters something that is not blank (valid input)
      while (true){
        String userInput = JOptionPane.showInputDialog(text);
        if (userInput == null) return null;                     // Capture cancelled
        if (!userInput.trim().isEmpty()) return userInput;
        JOptionPane.showMessageDialog(null, "This field cannot be empty.");
      }
    }

    // Method prompts user to enter the age restriction (age between 2 and 18) and captures it
    private int captureAgeRestriction(){
      while (true){
        String userInput = JOptionPane.showInputDialog("Enter the series age restriction:");
        if (userInput == null) return -1;                       // Capture cancelled
        // Calling helper method checkAgeRestriction that checks the age is between 2 and 18
        int age = checkAgeRestriction(userInput.trim());
        if (age != -1) return age;
        JOptionPane.showMessageDialog(null, "You have entered an incorrect series age restriction!!!\nPlease re-enter the series age restriction (2-18)");
      }
    }

    // Method prompts user to enter the number of episodes (number >=0) and captures it
    private int captureEpisodeCount(String seriesName){
      while (true){
        String userInput = JOptionPane.showInputDialog("Enter the number of episodes for " + seriesName);
        if (userInput == null) return -1;                       // Capture cancelled
        int episodeCount = checkPositiveInt(userInput.trim());
        if (episodeCount != -1) return episodeCount;
```

```
        JOptionPane.showMessageDialog(null, "Invalid input. Please enter a positive integer.");
    }
}

// HELPER METHODS FOR UPDATING SERIES
// Method confirms if the user want to update the series
private boolean confirmUpdate(SeriesModel series){
    // Confirming if the user wants to update the series
    int choice = JOptionPane.showConfirmDialog(
            null,
            "Would you like to proceed to updating this series?\n\n" + series.toString(),
            "Confirm Update",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE
    );
    return choice == JOptionPane.YES_OPTION;
}

// Method for updating the name of a captured TV series
private void updateName(SeriesModel series){
    // Confirming if the user wants to update the name
    int choice = JOptionPane.showConfirmDialog(
            null,
            "Do you want to update series name?",
            "Update Name",
            JOptionPane.YES_NO_OPTION
    );

    if (choice == JOptionPane.YES_OPTION){
        String newName = JOptionPane.showInputDialog(
                null,
                "Enter the new series name (Current series name: " + series.getSeriesName() + "): ",
                "Update Series Name",
                JOptionPane.QUESTION_MESSAGE
        );

        if (newName != null && !newName.trim().isEmpty()) {
            series.setSeriesName(newName.trim());
        }
    }
}

// Method for updating the age restriction of a captured TV series
private void updateAge(SeriesModel series){
    // Confirming if the user wants to update the age restriction
    int choice = JOptionPane.showConfirmDialog(
            null,
            "Do you want to update Age Restriction? (Current age restriction: " + series.getSeriesAge() + ")",
            "Update Age",
            JOptionPane.YES_NO_OPTION
    );

    if (choice == JOptionPane.YES_OPTION){
        int age = captureAgeRestriction();
        if (age != -1 ) {
            series.setSeriesAge(age);
        }
```

```
    }
  }

  // Method for updating the number of episodes of a captured TV series
  private void updateEpisodeCount(SeriesModel series){
  // Confirming if the user wants to update the age restriction
    int choice = JOptionPane.showConfirmDialog(
        null,
        "Do you want to update the number of episodes? (Current episode count: " +
series.getSeriesNumberOfEpisodes() + ")",
        "Update Number of Episodes",
        JOptionPane.YES_NO_OPTION
    );

    if (choice == JOptionPane.YES_OPTION){
      int episodes = captureEpisodeCount(series.getSeriesName());
      if (episodes != -1 ) {
        series.setSeriesNumberOfEpisodes(episodes);
      }
    }
  }

  // HELPER METHOD FOR DELETING SERIES
  private boolean confirmDelete(SeriesModel series){
    // Confirming if the user wants to delete the series
    int choice = JOptionPane.showConfirmDialog(
        null,
        "Are you sure you want to delete series " + series.getSeriesId() + " from the system?\n\n" +
series.toString(),
        "Confirm Delete",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.WARNING_MESSAGE
    );
    return choice == JOptionPane.YES_OPTION;
  }
/* ------------------------------------------------HELPER METHODS FOR UNIT TESTING-------------------------
------------------------- */
  // Capture series method that doesnt use dialogs (suitable for unit testing)
  void addSeriesForTest(SeriesModel series) {
    seriesList.add(series);
  }

  // Update series method that doesnt use dialogs (suitable for unit testing)
  boolean updateSeriesForTest(String id, String newName, int newAge, int newEpisodes) {
    SeriesModel series = searchByID(id);
    if (series == null) return false;

    if (newName != null && !newName.trim().isEmpty()) {          // Checks if name is not blank or null
      series.setSeriesName(newName.trim());
    }
    if (newAge >= 2 && newAge <= 18) {                    // Checks if age is witin range
      series.setSeriesAge(newAge);
    }
    if (newEpisodes >= 0) {
      series.setSeriesNumberOfEpisodes(newEpisodes);
    }
    return true;
```

```
    }

    // Delete series (delete by ID) method that doesnt use dialogs (suitable for unit testing)
    boolean deleteSeriesForTest(String id) {
        SeriesModel series = searchByID(id);
        return (series != null) && seriesList.remove(series);              // Method returns true if series is
deleted and false if series is not found
    }
}

// ------------------------------------------------------------------------------------------------------------------
--------- //
// END OF FILE                                                                                                      //
// ------------------------------------------------------------------------------------------------------------------
--------- //
```

Sample Code Output for Section A – Question 1

Starting Application and Selecting from Menu

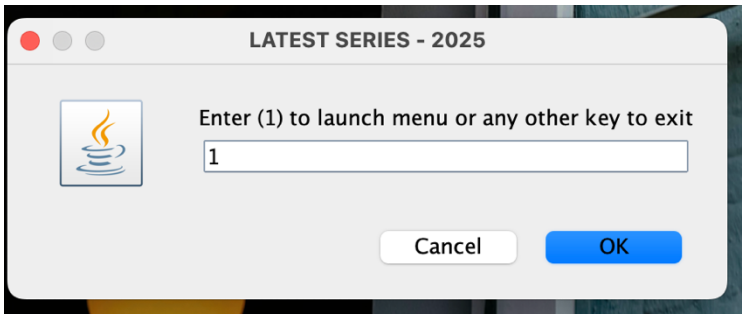

Figure 1: Starting application and launching menu

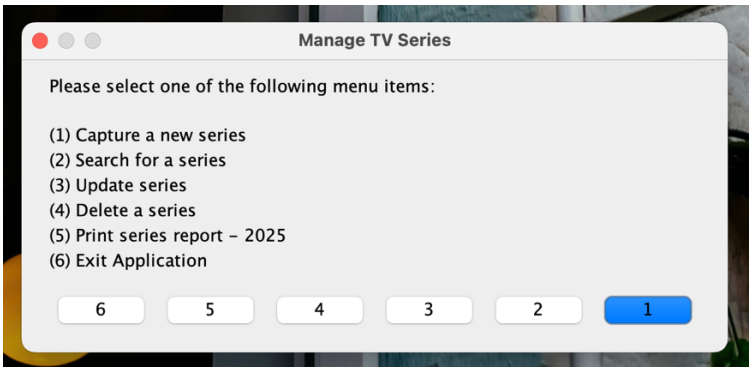

Figure 2: Displaying menu

Capturing a TV Series

Figure 3: User prompted to enter the series ID

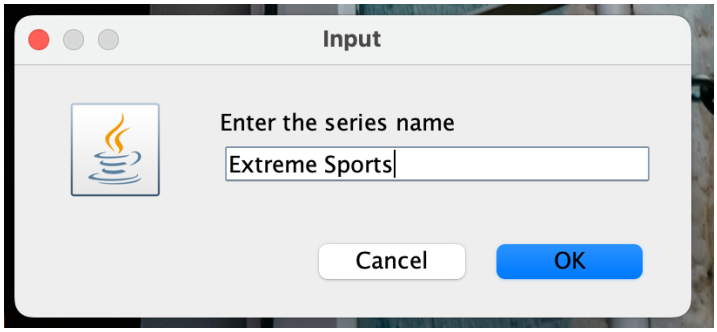

Figure 4: User prompted to enter the series name

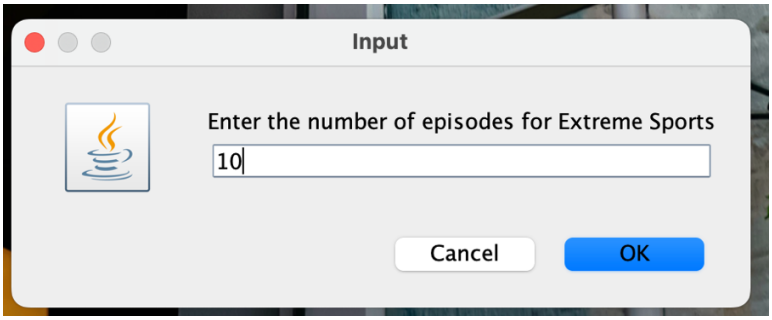

Figure 5: User prompted to enter the number of episodes in the series

Entering the Series Age Restriction

Figure 6: User prompted to enter the age restriction (valid input)

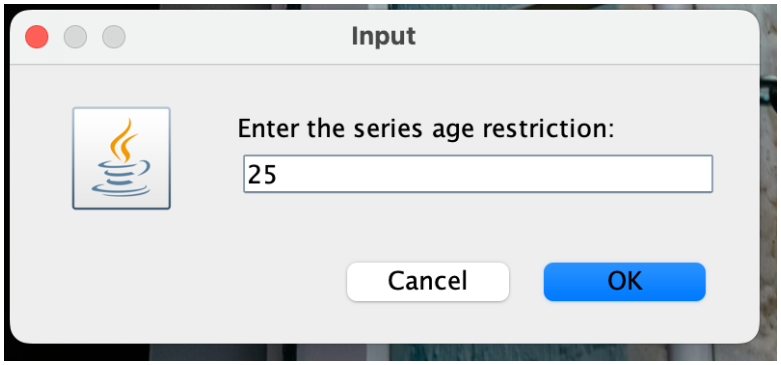

Figure 7: User prompted to enter the age restriction (invalid input)

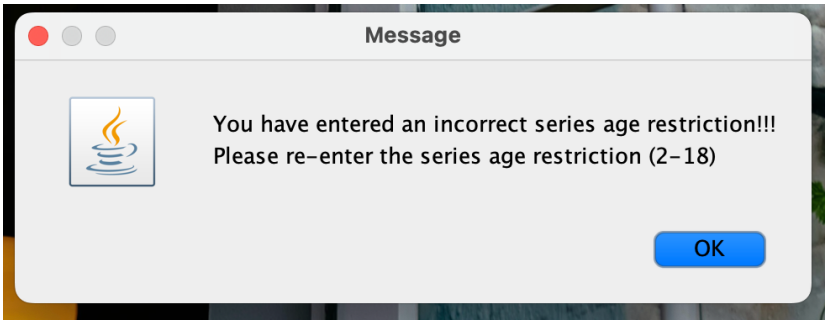

Figure 8: Error message is displayed due invalid input from the user

Confirming Successful Capture of TV Series

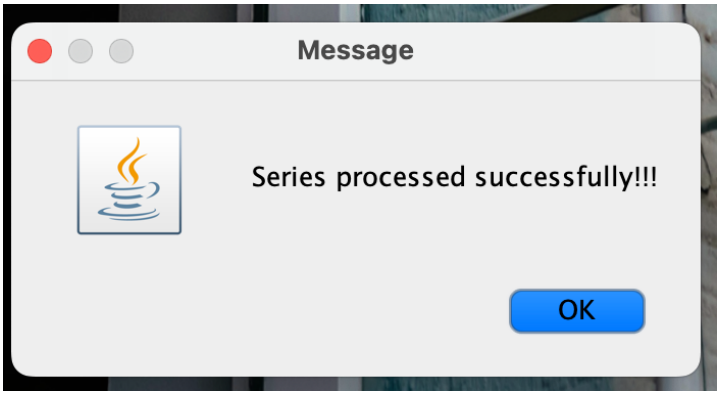

Figure 9: Confirmation of Successful Capture

Searching for a TV Series

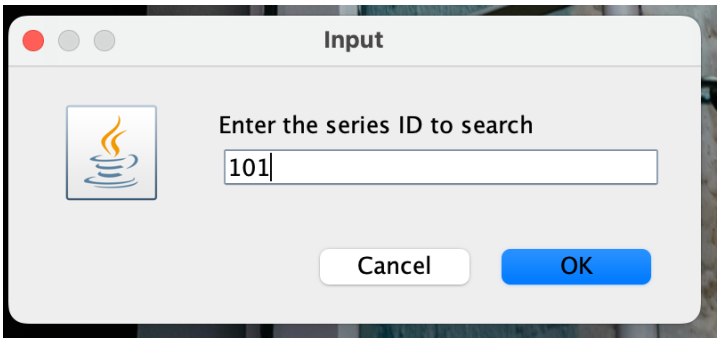

Figure 10: User prompted to enter the series ID to search (valid input, i.e. a series with ID 101 has been captured)

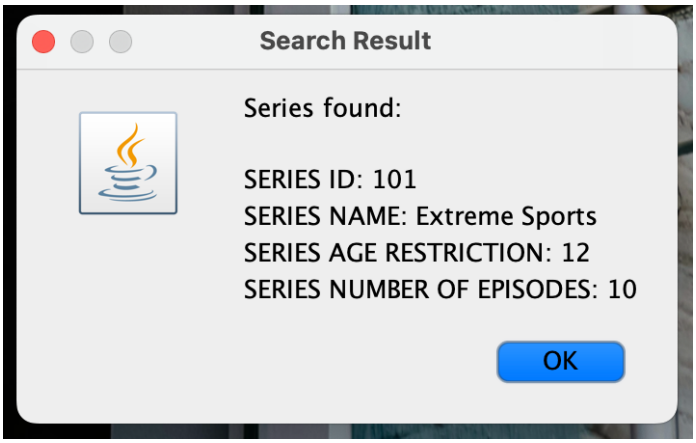

Figure 11TV series located successfully using series ID

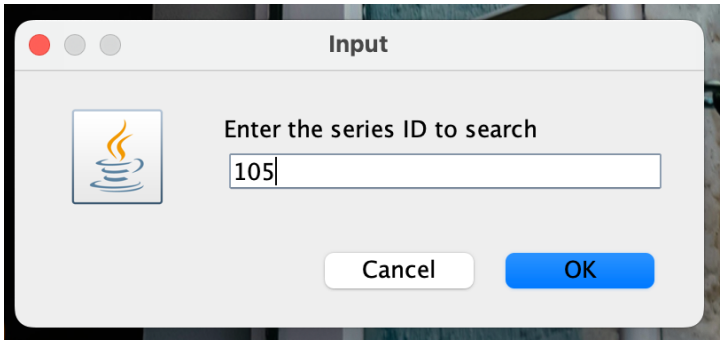

Figure 12: User prompted to enter the series ID to search (invalid input)

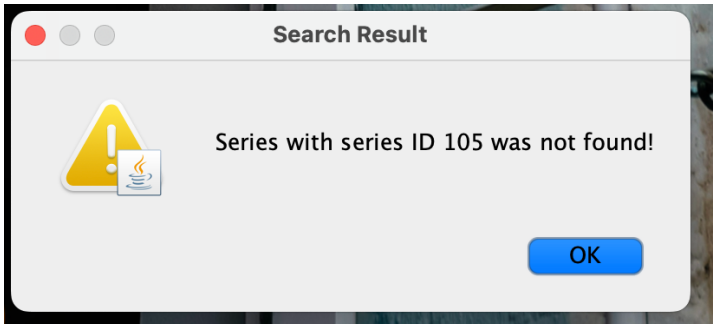

Figure 13: Error message is displayed (series with ID 105 could not be located)

Updating a TV Series

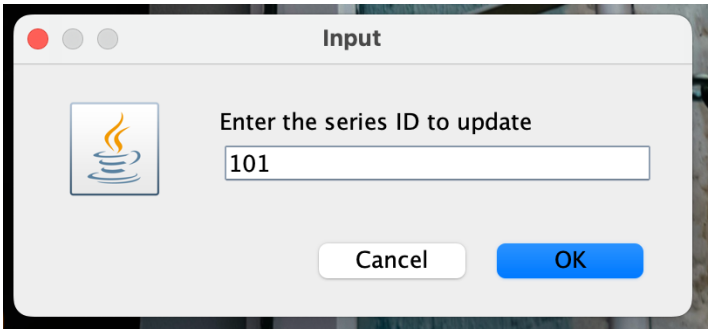

Figure 14: User prompted to enter the series ID to search

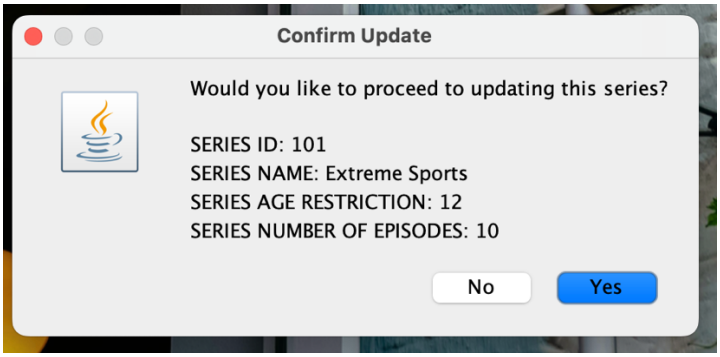

Figure 15: User asked to confirm if they would like to proceed to updating the series

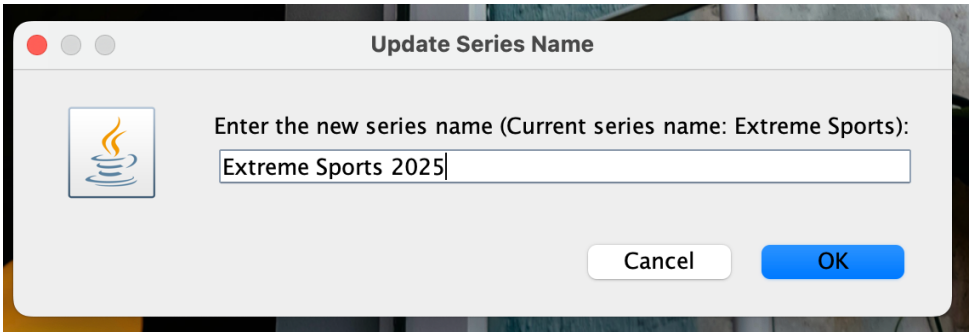

Figure 16: User prompted to enter the new series name

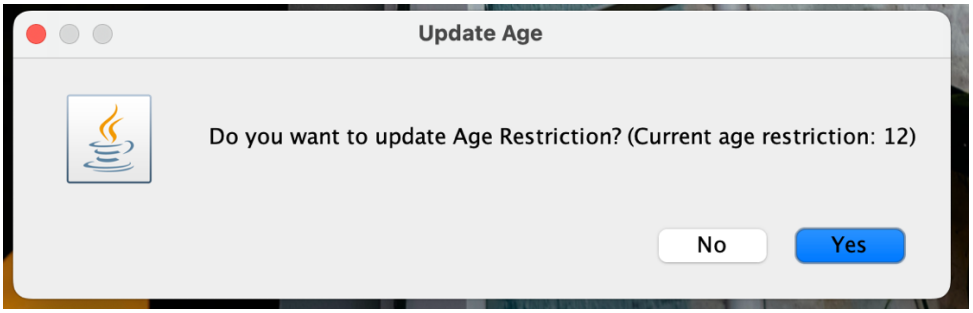

Figure 17: User asked to confirm if they would like to update the age restriction

Figure 18: User prompted to enter the new series age restriction

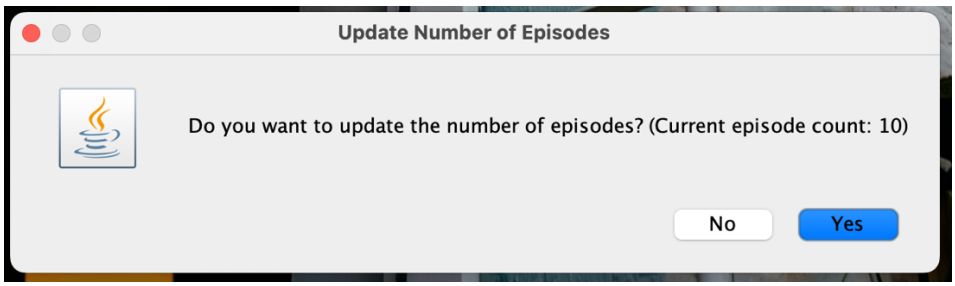

Figure 19: User asked to confirm if they would like to update the number of episodes

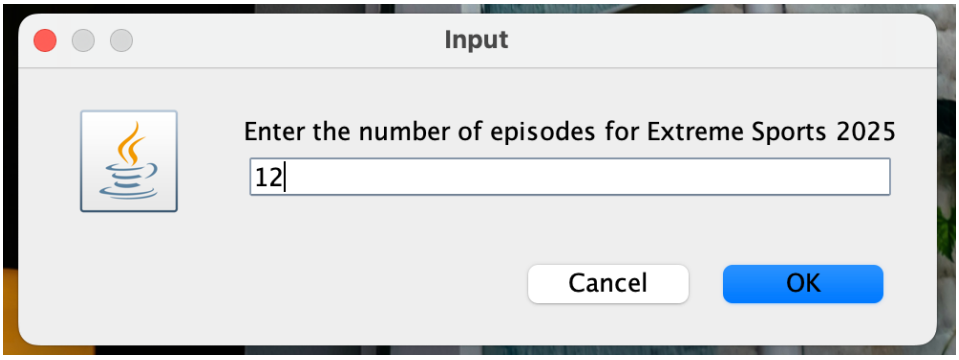

Figure 20: User prompted to enter the new number of episodes

Confirming Successful Update of TV Series

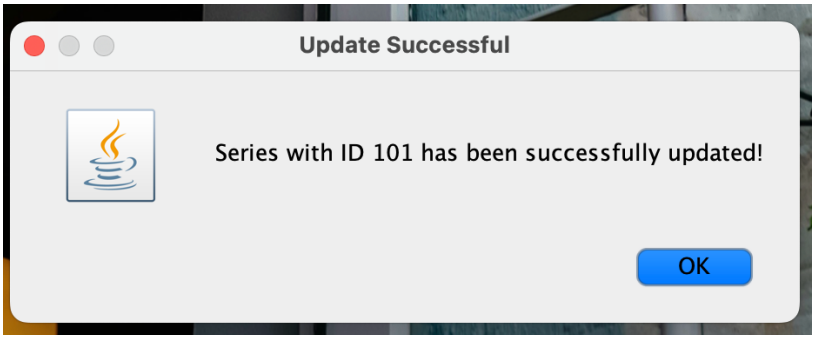

Figure 21: Confirmation that the series was updated successfully

Deleting a TV Series

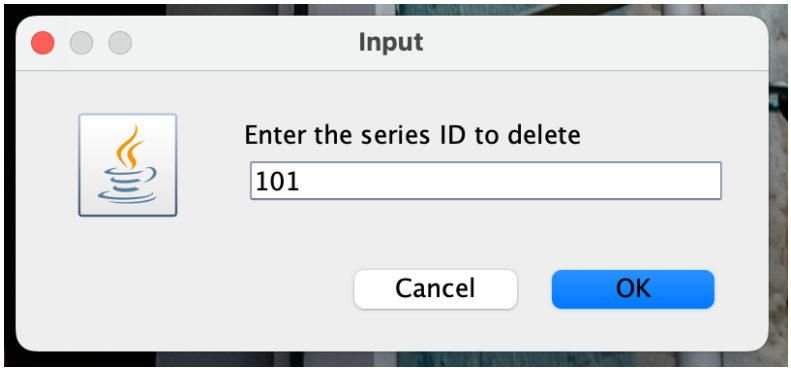

Figure 22: User prompted to enter the series ID to delete a series

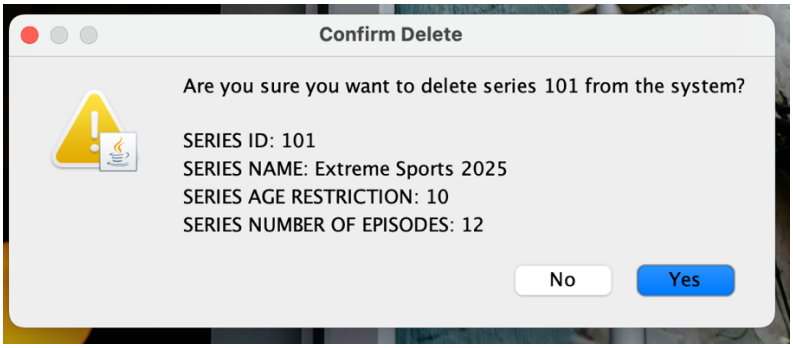

Figure 23: User asked to confirm if they would like to delete the series

Confirmation of Successful Series Deletion

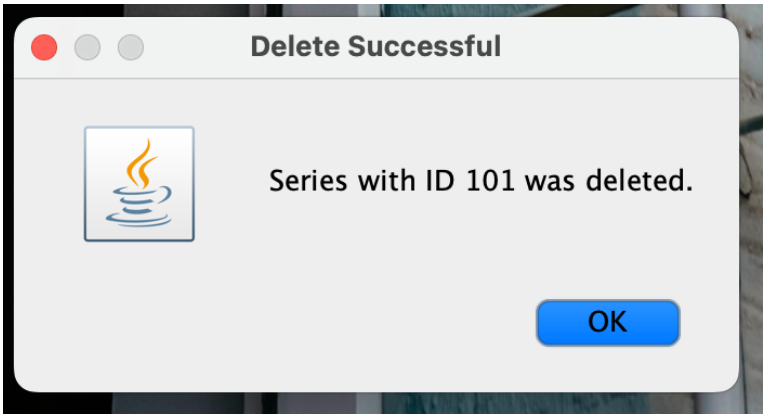

Figure 24: Confirmation that the series was deleted successfully

Displaying TV Series Report

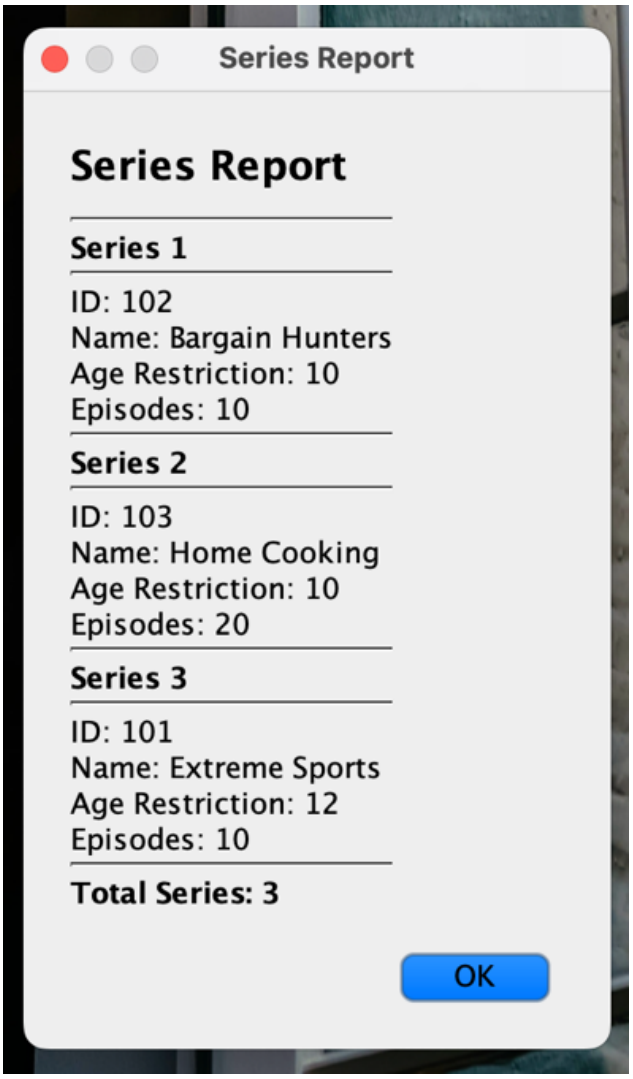

Figure 25: TV series report displayed successfully

Unit Testing

The following unit tests are located in the SeriesTest class.

Note: Please see Appendix A for the complete unit test code in SeriesTest.java

Table of Unit Test Data			
Attribute	Series 1	Series 2	Series 3
Series ID	A123	B456	C789
Series Name	“Modern Family”	“Stranger Things”	“Mickey Mouse”
Series Age Restriction	13	16	6
Series Number of Episodes	200	60	45

testSearchSeries()

```
// TEST 1
  @Test
  public void testSearchSeries() {
    SeriesModel result = series.searchByID("C789");
    assertNotNull(result);
    assertEquals("C789", result.getSeriesId());
    assertEquals("Mickey Mouse", result.getSeriesName());
    assertEquals(6, result.getSeriesAge());
    assertEquals(45, result.getSeriesNumberOfEpisodes());
  }
```

testSearchSeries_SeriesNotFound ()

```
// TEST 2
  @Test
  public void testSearchSeries_SeriesNotFound(){
    assertNull(series.searchByID("Z246"));
  }
```

testUpdateSeries()

```
// TEST 3
  @Test
  public void testUpdateSeries() {
    // Using unit test helper method in Series class
    boolean updatedSeries = series.updateSeriesForTest("B456", "Stranger Thingzz (Updated Name)",
17, 70);
    assertTrue(updatedSeries);

    SeriesModel result = series.searchByID("B456");
    assertNotNull(result);
    assertEquals("Stranger Thingzz (Updated Name)", result.getSeriesName());
    assertEquals(17, result.getSeriesAge());
    assertEquals(70, result.getSeriesNumberOfEpisodes());
  }
```

testDeleteSeries()

```
// TEST 4
  @Test
  public void testDeleteSeries() {
    // Using unit test helper method in Series class
    boolean deletedSeries = series.deleteSeriesForTest("A123");
    assertTrue(deletedSeries);
    assertNull(series.searchByID("A123"));
  }
```

TestDeleteSeries_SeriesNotFound()

```
// TEST 5
  @Test
  public void testDeleteSeries_SeriesNotFound(){
    // Using an ID that cannot be found/doesn't exist "Z246"
    boolean deletedSeries = series.deleteSeriesForTest("Z246");        // Using unit test helper method
in Series class
    assertFalse(deletedSeries);
    assertNotNull(series.searchByID("B456"));
  }
```

TestSeriesAgeRestriction_AgeValid()

```
// TEST 6
  @Test
  public void testSeriesAgeRestriction_AgeValid(){
    // Testing several values (ages) that lie within the allowed range (2-18): 2, 7, 10 and 18
    assertEquals(2, series.checkAgeRestriction("2"));
    assertEquals(7, series.checkAgeRestriction("7"));
    assertEquals(10, series.checkAgeRestriction("10"));
    assertEquals(18, series.checkAgeRestriction("18"));
  }
```

TestSeriesAgeRestriction_SeriesAgeInValid()

```
// TEST 7
  @Test
  public void testSeriesAgeRestriction_AgeInValid(){
    // Testing several values (ages) that are in the incorrect format or are not within the allowed age
restriction range (2-18)
    // The follwoing are not within the allowed age restriction range (2-18)
    assertEquals(-1, series.checkAgeRestriction("1"));
    assertEquals(-1, series.checkAgeRestriction("19"));
    // The following are not in the correct format
    assertEquals(-1, series.checkAgeRestriction("abc"));          // not a number
    assertEquals(-1, series.checkAgeRestriction(""));             // emptry
    assertEquals(-1, series.checkAgeRestriction("   "));          // whitespace
  }
```

Section B: Referencing Console Application

The following application enables users to generate reference lists comprising book and website references. It prompts the user to enter the details of each reference, such as authors, title, date accessed, publication year and place of publication.

Please note: for clarity and readability, I have changed the title blocks and comments from black to grey text.

Complete Code for Section B - Question 1

Main.java

```
/*
================================================================================================
================================================*/
/* Referencing Console Application
/*
================================================================================================
================================================*/
/* Name: Voningo Flora Makhubele Pecego
/* Student Number: ST10483982
/* Assignment: Practical Assignment - Section B (Question 1)
/* Lecturer: Amakan Elisha Agoni (BCA1 GRO3)
/* Date: 04/09/2025
/* ------------------------------------------------------------------------------------------------
--------*/
/* MAIN METHOD (Main.java)
/* ------------------------------------------------------------------------------------------------
--------*/

package referenceapplication;

import java.util.Scanner;

public class Main {
   // Creating a scanner object in order to read all user input from the console
   private static final Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
      // Creating manager object for storing and organising the references
      ReferenceManager manager = new ReferenceManager();

      // Loops in order to keep showing the menu until user enters (0) to exit
      while (true){
         System.out.println("\n================= Referencing Application =================");
         System.out.println("Remaining slots: " + manager.getRemainingSlots());  // Displaying the number
of slots that are remaining
         System.out.println("1) Add Book");
         System.out.println("2) Add Website");
         System.out.println("3) Print Reference List");
         System.out.println("0) Exit");
         System.out.print("Please choose an option: ");

         // Reading the user's menu choice
```

```
        String menuChoice = scanner.nextLine().trim();

        // Using switch statement to handle the menu choice
        switch (menuChoice){
          case "1" -> addBook(manager);                    // adding book reference
          case "2" -> addWebsite(manager);                 // adding website reference
          case "3" -> displayReferenceList(manager);       // showing all the stored references
          case "0" -> {
            System.out.println("Until next time!");
            return;                                        // exiting the application
          }
          default -> System.out.println("Invalid option. Please enter 0, 1, 2, or 3.");
        }
      }
    }

    // Method prompts user to enter the book details and adds the book reference to ReferenceManager
if there is space available
    private static void addBook(ReferenceManager manager){
      // Checking if the space is full (max 50 references)
      if(manager.arrayIsFull()){
        System.out.println("List is full. Please print the list or exit the application.");
        return;
      }

      System.out.println("\n----------------- Add Book -----------------");
      String[] authors = captureAuthors();
      int year = checkIntegerInput("Year: ", true);
      String title = checkInput("Title: ", true);
      String placeOfPublication = checkInput("Place/City of Publication: ", true);
      String publisher = checkInput("Publisher: ", true);
      String edition = checkInput("Edition (e.g. 2nd ed.) [leave blank for 1st edition]: ", false);

      // Creating a book object and adding it to the reference manager
      Book newBook = new Book(authors, year, title, placeOfPublication, publisher, edition);
      boolean isAdded = manager.addNewReference(newBook);
      System.out.println(isAdded ? "Book reference added successfully!" : "Capacity is full. Could not
add reference.");
    }

    // Method prompts user to enter the website details and adds the website reference to
ReferenceManager if there is space available
    private static void addWebsite(ReferenceManager manager){
      // Checking if the space is full (max 50 references)
      if (manager.arrayIsFull()){
        System.out.println("Capacity is full. Please print the list or exit the application");
        return;
      }

      System.out.println("\n----------------- Add Website -----------------");
      String[] authors = captureAuthors();
      int year = checkIntegerInput("Year: ", true);
      String title = checkInput("Page Title: ", true);
      String website = checkInput("Website Name: ", true);
      String url = checkInput("URL: ", true);
      String dateAccessed = checkInput("Date Accessed (e.g. 3 September 2025): ", false);
```

```
      // Creating a website object and adding it to the reference manager
      Website newWebsite = new Website(authors, year, title, website, url, dateAccessed);
      boolean isAdded = manager.addNewReference(newWebsite);
      System.out.println(isAdded ? "Website reference added successfully!" : "Capacity is full. Could not
add reference.");
    }

    // Method sorts and displays all of the stored references in a numbered list
    private static void displayReferenceList(ReferenceManager manager){
      System.out.println("\n================= LIST OF REFERENCES =================");
      // Sorting the references before printing them
      manager.sortByAuthorThenYear();
      Reference[] allReferences = manager.getAllReferences();

      if (allReferences.length == 0){
        // Displaying an error message if no references exitst yet
        System.out.println("No references exist yet. Please add a Book or Website reference first.");
        return;
      }

      // Printing each reference
      for (Reference reference : allReferences) {
        System.out.println(reference.formatReference());
        System.out.println();                          // Adding an extra blank line between references
      }
      System.out.println("-----------------------------------------------------");
      System.out.println("Total references: " + allReferences.length);
    }

    // USER INPUT METHODS
    private static String checkInput(String label, boolean isRequired){
      while (true){
        System.out.print(label);
        String value = scanner.nextLine().trim();

        if (!isRequired) return value;
        // Requres that the input is non-blank
        if (!value.isBlank()) return value;

        System.out.println("This field cannot be empty.");
      }
    }

    private static int checkIntegerInput(String label, boolean isPositive){
      while (true){
        System.out.print(label);
        String textInput = scanner.nextLine().trim();
        // Exception Handling
        try{
          int input = Integer.parseInt(textInput);

          if (isPositive && input < 0){
            System.out.println("Please enter 0 or a positive number.");
            continue;
          }
          return input;
        } catch (NumberFormatException e){
```

```
            System.out.println("Please enter a valid number.");
        }
    }
  }

  private static String formatInitials(String authorNames){
    String formatted;
    if (authorNames == null){
      formatted = "";
    } else {
      formatted = authorNames.trim();
    }

    if (formatted.isEmpty()) return "";

    StringBuilder initials = new StringBuilder();
    String[] parts = formatted.split("\\s+");                    // Regex obtained from ChatGPT (2025)
    for (String section : parts){
      initials.append(Character.toUpperCase(section.charAt(0))).append('.');
    }
    return initials.toString();

  }

  // BOOK REFERENCING METHODS
  private static String[] captureAuthors(){
    int authorCount = checkIntegerInput("Number of authors (0 for no author): ", true);
    if (authorCount == 0) return new String[0];

    String[] authors = new String[authorCount];
    for (int index = 0; index < authorCount; index++){
      System.out.println("Author " + (index + 1));
      String names = checkInput("Name(s): ", true);
      String surname    = checkInput("Surname: ", true);

      String initials = formatInitials(names);
      authors[index] = surname.trim() + ", " + initials;
    }
    return authors;
  }

}

// ------------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                          //
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

Reference.java

```
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* REFERENCE CLASS (Reference.java)                                                                                      /
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
```

```
package referenceapplication;

public abstract class Reference {
   private final String[] authors;
   private final int publicationYear;
   private final String title;

   // CONSTRUCTOR
   public Reference(String[] authors, int publicationYear, String title){
      // for authors
      if (authors != null) {
         this.authors = new String[authors.length];

         for (int index = 0; index < authors.length; index++){
            this.authors[index] = authors[index];
         }
      } else{
         this.authors = new String[0];
      }
      // for publication year
      this.publicationYear = publicationYear;

      // for title
      if (title != null){
         this.title = title.trim();
      } else {
         this.title = "";
      }
   }

   // ABSTRACT METHOD
   // Method forces all subclasses to define how their references are formatted
   public abstract String formatReference();

   // OTHER METHODS
   public String [] getAuthors(){
      String[] authorsCopy = new String[authors.length];
      for (int index = 0; index < authors.length; index++){
         authorsCopy[index] = authors[index];
      }
      return authorsCopy;
   }

   public int getPublicationYear(){
      return publicationYear;
   }

   public String getTitle(){
      return title;
   }

   public String findSortingAuthorKey(){
      // If there are no authors (i.e. authors.length == 0), return "~" since "~" character comes after all
normal letters in Unicode/ASCII order (ChatGPT, 2025)
      if (authors.length == 0) return "~";

      // Declaring variables
```

```
    String firstAuthor = authors[0];                    // The first author in the list
    String surname;

    // Finding position of the comma (indexOf locates the character in the string and returns its
position)
    int commaIndex = firstAuthor.indexOf(',');                  // If the comma is found the position is
retuned as an integer (i.e. commaIndex); if comma not found then returns -1

    if (commaIndex > 0){                                // If there is a comma, take everything before it, i.e.
the surname
      surname = firstAuthor.substring(0, commaIndex);
    } else {
      surname = firstAuthor;                            // If no comma then take the whole string
    }
    return surname.trim().toLowerCase();
  }

  public static String formatAuthors(String[] authors) {
    if (authors == null || authors.length == 0) return "";
    if (authors.length == 1) return authors[0];                 // One author
    if (authors.length == 2) return authors[0] + " and " + authors[1];    // Two authors

    // More than 2 authors
    StringBuilder formattedAuthors = new StringBuilder();
    for (int index = 0; index < authors.length; index++){
      if (index > 0){
        formattedAuthors.append(index == authors.length - 1 ? " and " : ", ");
      }
      formattedAuthors.append(authors[index]);
    }
    return formattedAuthors.toString();
  }

}

// ------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                          //
// ------------------------------------------------------------------------------------------------------------------
---------- //
```

Book.java

```
/* ------------------------------------------------------------------------------------------------------------------
--------*/
/* BOOK CLASS (Book.java)                                                                                /
/* ------------------------------------------------------------------------------------------------------------------
--------*/

package referenceapplication;

public class Book extends Reference {
  private final String placeOfPublication;
  private final String publisher;
  private final String edition;
```

```
  // CONSTRUCTOR
  public Book(String[] authors, int publicationYear, String title, String placeOfPublication, String
publisher, String edition){
    super(authors, publicationYear, title);

    // for place of publication
    if(placeOfPublication != null){
      this.placeOfPublication = placeOfPublication.trim();
    } else {
      this.placeOfPublication = "";
    }

    // for publisher
    if (publisher != null){
      this.publisher = publisher.trim();
    } else {
      this.publisher = "";
    }

    // for edition
    if (edition != null){
      this.edition = edition.trim();
    } else {
      this.edition = "";
    }
  }

  @Override
  public String formatReference(){
    boolean includeEdition = !edition.isBlank() && !edition.equalsIgnoreCase("1st ed.") &&
!edition.equals("1");

    String editionText;

    // Edition included in reference
    if (includeEdition){
      editionText = " " + edition;
    }
    else {
      editionText = "";
    }

    String formattedRef = String.format("%s %d. %s.%s %s: %s.",
        Reference.formatAuthors(getAuthors()),
        getPublicationYear(),
        getTitle(),
        editionText,
        placeOfPublication,
        publisher
    );

    // Line reccommended by ChatGPT (2025) to clean up accidental double dots
    return formattedRef.replace("..", ".").trim();
  }
}
```

```
// ------------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                          //
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

Website.java

```
/* ------------------------------------------------------------------------------------------------------------------------
--------*/
/* WEBSITE CLASS (Webstie.java)                                                                                          /
/* ------------------------------------------------------------------------------------------------------------------------
--------*/

package referenceapplication;

public class Website extends Reference {
   private final String websiteName;
   private final String url;
   private final String dateAccessed;

   // CONSTRUCTOR
   public Website(String[] authors, int publicationYear, String title, String websiteName, String url, String
dateAccessed){
      super(authors, publicationYear, title);

      // for website name
      if (websiteName != null){
         this.websiteName = websiteName.trim();
      } else {
         this.websiteName = "";
      }

      // for url
      if (url != null){
         this.url = url.trim();
      } else {
         this.url = "";
      }

      // for date accessed
      if (dateAccessed != null){
         this.dateAccessed = dateAccessed.trim();
      } else {
         this.dateAccessed = "";
      }
   }

   @Override
   public String formatReference(){
      String accessedText = dateAccessed.isBlank() ? "" : " [Accessed " + dateAccessed + "]";
      return String.format("%s %d. %s. %s. Available at: %s%s.",
           Reference.formatAuthors(getAuthors()),
           getPublicationYear(),
           getTitle(),
           websiteName,
```

```
            url,
            accessedText
        ).trim();
    }
}

// ------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                      //
// ------------------------------------------------------------------------------------------------------------------
---------- //
```

ReferenceManager.java

```
/* ------------------------------------------------------------------------------------------------------------------
--------*/
/* REFERENCE MANAGER CLASS (ReferenceManager.java)                                                                  /
/* ------------------------------------------------------------------------------------------------------------------
--------*/

package referenceapplication;

public class ReferenceManager {
    // Setting the fixed maximum array size/capacity (50 references)
    private static final int MAX_REFERENCES = 50;

    // Declaring the array of references for storage (empty slotes are initially null)
    private final Reference[] references = new Reference[MAX_REFERENCES];

    // Number of references stored currently
    private int refCount = 0;

    // Methods checks if there is space (i.e maximum references not reached) and adds the reference to
the next free slot
    public boolean addNewReference(Reference ref){
        // Returns false if capacity is full or null
        if (ref == null) return false;                          // protecting agains null inputs
        if (arrayIsFull()) return false;
        references[refCount++] = ref;
        return true;                                    // Returns true if reference was added
    }

    // BUBBLE SORT (With early exit optimization using a swap flag)
    public void sortByAuthorThenYear(){
        for (int pass = 0; pass < refCount -1; pass++){
            boolean swapped = false;

            for (int position = 0; position < refCount - pass - 1; position++){
                // Calling helper method compareReferences(refA, refB)
                if (compareReferences(references[position], references[position + 1]) > 0){
                    Reference temp = references[position];
                    references[position] = references[position + 1];
                    references[position + 1] = temp;
                    swapped = true;
                }
            }
```

```
            // already sorted (i.e no swaps occured in the entire pass) therefore early exit
            if (!swapped) return;
        }
    }

    // Method for returning a trimmed copy of the stored references
    public Reference[] getAllReferences(){
        // Creating a new array sized exaclty to refCount
        Reference[] result = new Reference[refCount];
        for (int index = 0; index < refCount; index++){
            // Copying each used element
            result[index] = references[index];
        }
        return result;
    }

    // Method used to define the sort order (alphabetical then numberical)
    private int compareReferences(Reference refA, Reference refB){
        // Comparing the authors
        int authorComparison = refA.findSortingAuthorKey().compareTo(refB.findSortingAuthorKey());
        if (authorComparison != 0) return authorComparison;
        // Comparing the publication year
        return Integer.compare(refA.getPublicationYear(), refB.getPublicationYear());
    }

    // HELPER AND GETTER METHODS
    public boolean arrayIsFull(){
        return refCount >= references.length;
    }

    public int getRefCount(){
        return refCount;
    }

    public int getRemainingSlots(){
        return references.length - refCount;
    }

}

// ------------------------------------------------------------------------------------------------------------------------
---------- //
// END OF FILE                                                                                                            //
// ------------------------------------------------------------------------------------------------------------------------
---------- //
```

Sample Code Output for Section B – Question 1

Capturing a Book Reference

```
================= Referencing Application =================
Remaining slots: 50
1) Add Book
2) Add Website
3) Print Reference List
0) Exit

Please choose an option: 1

----------------- Add Book -----------------
Number of authors (0 for no author): 2
Author 1
Name(s): James
Surname: Smith
Author 2
Name(s): Samantha
Surname: Anderson
Year: 2023
Title: Introduction to Programming
Place/City of Publication: London
Publisher: Green Publishing Group
Edition (e.g. 2nd ed.) [leave blank for 1st edition]: 3rd ed.
Book reference added successfully!
```

Figure 26: Reference 1 (50 Slots Remaining Before Capture)

Capturing a Website Reference

```
================= Referencing Application =================
Remaining slots: 49
1) Add Book
2) Add Website
3) Print Reference List
0) Exit

Please choose an option: 2

----------------- Add Website -----------------
Number of authors (0 for no author): 3
Author 1
Name(s): Sam
Surname: Sanders
Author 2
Name(s): Elizabeth
Surname: Brown
Author 3
Name(s): Eli
Surname: Webster
Year: 1999
Page Title: Top 10 Artists of 1999
Website Name: Music Insider
URL: https://www.musicinsider.com
Date Accessed (e.g. 3 September 2025): 2 August 2024
Website reference added successfully!
```

Figure 27: Reference 2 (49 Slots Remaining Before Capture)

Capturing a Book Reference with No Edition

```
================= Referencing Application =================
Remaining slots: 48
1) Add Book
2) Add Website
3) Print Reference List
0) Exit

Please choose an option: 1

----------------- Add Book -----------------
Number of authors (0 for no author): 1
Author 1
Name(s): Tyler
Surname: Pearson
Year: 2000
Title: Applied Mathematics for Beginners
Place/City of Publication: New York
Publisher: Happy Publishers
Edition (e.g. 2nd ed.) [leave blank for 1st edition]:
Book reference added successfully!
```

Figure 28: Reference 3 (48 Slots Remaining Before Capture)

Displaying the List of References

```
================ Referencing Application ================
Remaining slots: 47
1) Add Book
2) Add Website
3) Print Reference List
0) Exit

Please choose an option: 3

================ LIST OF REFERENCES ================
Pearson, T. 2000. Applied Mathematics for Beginners. New York: Happy Publishers.

Sanders, S., Brown, E. and Webster, E. 1999. Top 10 Artists of 1999. Music Insider. Available at: https://www.musicinsider.com [Accessed 2 August 2024].

Smith, J. and Anderson, S. 2023. Introduction to Programming. 3rd ed. London: Green Publishing Group.

------------------------------------------------------
Total references: 3
```

Figure 29: Selecting Option (3) to Display List of References

Exiting Application

```
================== Referencing Application ==================
Remaining slots: 47
1) Add Book
2) Add Website
3) Print Reference List
0) Exit

Please choose an option: 0
```

Figure 30: 47 Remaining Slots Before Exit (i.e. 3 References Captured)

Unit Testing

Below are the *main unit tests* created for the application, apart from the test stubs that are automatically generated by Netbeans.

Additionally**, the complete unit test code in each test file can be viewed using the GitHub link**

Book.java Tests

```
    // TEST 1 ------------------------------------------------------------------
    @Test
    public void testFormatReferenceWithEdition() {
        Book bookInput = new Book(
                new String[]{"Smith, J.", "Anderson, A."},
                1990,
                "Java for Beginners",
                "Cape Town",
                "Green Publishing Group",
                "2nd ed."
        );

        String ref = bookInput.formatReference();

        assertTrue(ref.contains("Smith, J. and Anderson, A."));
        assertTrue(ref.contains(" 1990. "));
        assertTrue(ref.contains("Java for Beginners"));
        assertTrue(ref.contains(" 2nd ed."));
        assertTrue(ref.contains(" Cape Town: Green Publishing Group"));
    }

    // TEST 2 ------------------------------------------------------------------
    @Test
    public void testFormatReferenceNoEdition() {
        Book bookInput = new Book(new String[]{"Sanders, Z."}, 2022, "Dictionary", "London", "Publisher 1",
"");
        String ref = bookInput.formatReference();
        assertFalse(ref.toLowerCase().contains("ed."));
    }

    // TEST 3 ------------------------------------------------------------------
    @Test
    public void testFormatReferenceFirstEdition(){
        Book bookInput = new Book(new String[]{"Sanders, Z."}, 2022, "Dictionary", "London", "Publisher 1",
"1st ed.");
        String ref = bookInput.formatReference();
        assertFalse(ref.contains("1st ed."));
    }
```

Website.java Tests

```
    // TEST 1 ------------------------------------------------------------------
    @Test
```

```
    public void testFormatReferenceInclDate() {
        Website websiteInput = new Website(
                new String[]{"Bruce, W."},
                2023,
                "Top 10 Songs 2023",
                "Music Direct",
                "https://www.website1.com",
                "3 September 2025"
        );

        String ref = websiteInput.formatReference();

        assertTrue(ref.contains("Bruce, W."));
        assertTrue(ref.contains(" 2023. "));
        assertTrue(ref.contains("Top 10 Songs 2023"));
        assertTrue(ref.contains("Music Direct"));
        assertTrue(ref.contains(" Available at: https://www.website1.com"));
        assertTrue(ref.contains("[Accessed 3 September 2025]"));
    }

    // TEST 2 ------------------------------------------------------------------
    @Test
    public void testFormatReferenceNoDate() {
        Website websiteInput = new Website(
                new String[]{"Bruce, W."},
                2023,
                "Top 10 Songs 2023",
                "Music Direct",
                "https://www.website1.com",
                ""
        );

        String ref = websiteInput.formatReference();

        assertFalse(ref.contains("Accessed"));
        assertTrue(ref.contains("Available at: https://www.website1.com"));
    }
```

ReferenceManager.java Tests

```
// TEST 1 ------------------------------------------------------------------
    @Test
    public void testAddOneNewReference() {
        ReferenceManager manager = new ReferenceManager();
        Book book = new Book(new String[]{"Smith, J."}, 1990, "Java for Beginners", "Cape Town", "Green
Publishing Group", "3rd ed.");

        boolean isAdded = manager.addNewReference(book);

        assertTrue(isAdded);
        assertEquals(1, manager.getRefCount());
        assertEquals(49, manager.getRemainingSlots());
        assertEquals(1, manager.getAllReferences().length);
    }
```

```
  // TEST 2 ------------------------------------------------------------------
  @Test
  public void addNewReferenceNullInput() {
    ReferenceManager manager = new ReferenceManager();
    boolean isAdded = manager.addNewReference(null);
    assertFalse(isAdded);
    assertEquals(0, manager.getRefCount());
  }

  // TEST 3 ------------------------------------------------------------------
  @Test
  public void sortByAuthorThenYear() {
    ReferenceManager manager = new ReferenceManager();

    manager.addNewReference(new Book(new String[]{"Sanders, Z."}, 2022, "Dictionary", "London",
"Publisher 1", "5th ed."));
    manager.addNewReference(new Book(new String[]{"West, A."}, 2021, "Novel", "Johannesburg",
"Publisher 2", "7th ed."));
    manager.addNewReference(new Book(new String[]{"Anderson, A."}, 2019, "Textbook", "New York",
"Publisher 3", "2 ed."));

    manager.sortByAuthorThenYear();
    Reference[] list = manager.getAllReferences();

    // Expected result: Anderson(2019), Sanders(2022), West(2021)
    assertEquals("Anderson, A.", list[0].getAuthors()[0]);
    assertEquals(2019, list[0].getPublicationYear());

    assertEquals("Sanders, Z.", list[1].getAuthors()[0]);
    assertEquals(2022, list[1].getPublicationYear());

    assertEquals("West, A.", list[2].getAuthors()[0]);
    assertEquals(2021, list[2].getPublicationYear());
  }
```

Appendices

Appendix A: Complete Unit Test Code in SeriesTest.java (TV Series Management Application)

Please note: for clarity and readibility I have changed the comments and title blocks from black to grey text.

```
/* ------------------------------------------------------------------------------------------------------------------
--------*/
/* UNIT TESTS FOR SERIES CLASS                                                                                      */
/* ------------------------------------------------------------------------------------------------------------------
--------*/

package tvseriesapplication;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;

public class SeriesTest {

    private Series series;

    public SeriesTest() {
    }

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Before
    public void setUp() {
        series = new Series();

        series.addSeriesForTest(new SeriesModel("A123", "Modern Family", 13, 200));
        series.addSeriesForTest(new SeriesModel("B456", "Stranger Things", 16, 60));
        series.addSeriesForTest(new SeriesModel("C789", "Mickey Mouse", 6, 45));
    }

    @After
    public void tearDown() {
    }
/*
 * ------------------------------------------------------------------------------------------------------------------
--------
 * MANDATORY UNIT TESTS
```

```
 * ---------------------------------------------------------------------------------------------------------------------------
--------
 * 1. testSearchSeries()
 * 2. testSearchSeries_SeriesNotFound()
 * 3. testUpdateSeries()
 * 4. testDeleteSeries()
 * 5. testDeleteSeries_SeriesNotFound()
 * 6. testSeriesAgeRestriction_AgeValid()
 * 7. TestSeriesAgeRestriction_SeriesAgeInValid()
 * ---------------------------------------------------------------------------------------------------------------------------
--------
 */

   // TEST 1
   @Test
   public void testSearchSeries() {
      SeriesModel result = series.searchByID("C789");
      assertNotNull(result);
      assertEquals("C789", result.getSeriesId());
      assertEquals("Mickey Mouse", result.getSeriesName());
      assertEquals(6, result.getSeriesAge());
      assertEquals(45, result.getSeriesNumberOfEpisodes());
   }

   // TEST 2
   @Test
   public void testSearchSeries_SeriesNotFound(){
      assertNull(series.searchByID("Z246"));
   }

   // TEST 3
   @Test
   public void testUpdateSeries() {
      // Using unit test helper method in Series class
      boolean updatedSeries = series.updateSeriesForTest("B456", "Stranger Thingzz (Updated Name)",
17, 70);
      assertTrue(updatedSeries);

      SeriesModel result = series.searchByID("B456");
      assertNotNull(result);
      assertEquals("Stranger Thingzz (Updated Name)", result.getSeriesName());
      assertEquals(17, result.getSeriesAge());
      assertEquals(70, result.getSeriesNumberOfEpisodes());
   }

   // TEST 4
   @Test
   public void testDeleteSeries() {
      // Using unit test helper method in Series class
      boolean deletedSeries = series.deleteSeriesForTest("A123");
      assertTrue(deletedSeries);
      assertNull(series.searchByID("A123"));
   }

   // TEST 5
   @Test
   public void testDeleteSeries_SeriesNotFound(){
```

```
    // Using an ID that cannot be found/doesn't exist "Z246"
    boolean deletedSeries = series.deleteSeriesForTest("Z246");          // Using unit test helper method
in Series class
    assertFalse(deletedSeries);
    assertNotNull(series.searchByID("B456"));
  }

  // TEST 6
  @Test
  public void testSeriesAgeRestriction_AgeValid(){
    // Testing several values (ages) that lie within the allowed range (2-18): 2, 7, 10 and 18
    assertEquals(2, series.checkAgeRestriction("2"));
    assertEquals(7, series.checkAgeRestriction("7"));
    assertEquals(10, series.checkAgeRestriction("10"));
    assertEquals(18, series.checkAgeRestriction("18"));
  }

  // TEST 7
  @Test
  public void testSeriesAgeRestriction_AgeInValid(){
    // Testing several values (ages) that are in the incorrect format or are not within the allowed age
restriction range (2-18)
    // The follwoing are not within the allowed age restriction range (2-18)
    assertEquals(-1, series.checkAgeRestriction("1"));
    assertEquals(-1, series.checkAgeRestriction("19"));
    // The following are not in the correct format
    assertEquals(-1, series.checkAgeRestriction("abc"));              // not a number
    assertEquals(-1, series.checkAgeRestriction(""));                 // emptry
    assertEquals(-1, series.checkAgeRestriction("   "));              // whitespace
  }


/*
 * ---------------------------------------------------------------------------------------------------------------
--------
 * AUTO-GENERATED TEST STUBS (NETBEANS PROTOTYPES)
 * ---------------------------------------------------------------------------------------------------------------
--------
 * 1. testCaptureSeries()
 * 2. testSeriesReport()
 * 3. testExitSeriesApplication()
 * 4. testCheckPositiveInt()
 * 5. testCheckAgeRestriction()
 * 6. testAddSeriesForTest()
 * 7. testUpdateSeriesForTest()
 * 8. testDeleteSeriesForTest()
 * 9. testSearchByID()
 * ---------------------------------------------------------------------------------------------------------------
--------
 */

  /**
   * Test of captureSeries method, of class Series.
   */
  @Test
  public void testCaptureSeries() {
    System.out.println("captureSeries");
```

```
    Series instance = new Series();
    instance.captureSeries();
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of seriesReport method, of class Series.
 */
@Test
public void testSeriesReport() {
    System.out.println("seriesReport");
    Series instance = new Series();
    instance.seriesReport();
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of exitSeriesApplication method, of class Series.
 */
@Test
public void testExitSeriesApplication() {
    System.out.println("exitSeriesApplication");
    Series instance = new Series();
    instance.exitSeriesApplication();
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of checkPositiveInt method, of class Series.
 */
@Test
public void testCheckPositiveInt() {
    System.out.println("checkPositiveInt");
    String userInput = "";
    Series instance = new Series();
    int expResult = 0;
    int result = instance.checkPositiveInt(userInput);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of checkAgeRestriction method, of class Series.
 */
@Test
public void testCheckAgeRestriction() {
    System.out.println("checkAgeRestriction");
    String userInput = "";
    Series instance = new Series();
    int expResult = 0;
    int result = instance.checkAgeRestriction(userInput);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
```

```
    fail("The test case is a prototype.");
}

/**
 * Test of addSeriesForTest method, of class Series.
 */
@Test
public void testAddSeriesForTest() {
    System.out.println("addSeriesForTest");
    SeriesModel series = null;
    Series instance = new Series();
    instance.addSeriesForTest(series);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of updateSeriesForTest method, of class Series.
 */
@Test
public void testUpdateSeriesForTest() {
    System.out.println("updateSeriesForTest");
    String id = "";
    String newName = "";
    int newAge = 0;
    int newEpisodes = 0;
    Series instance = new Series();
    boolean expResult = false;
    boolean result = instance.updateSeriesForTest(id, newName, newAge, newEpisodes);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of deleteSeriesForTest method, of class Series.
 */
@Test
public void testDeleteSeriesForTest() {
    System.out.println("deleteSeriesForTest");
    String id = "";
    Series instance = new Series();
    boolean expResult = false;
    boolean result = instance.deleteSeriesForTest(id);
    assertEquals(expResult, result);
    // TODO review the generated test code and remove the default call to fail.
    fail("The test case is a prototype.");
}

/**
 * Test of searchByID method, of class Series.
 */
@Test
public void testSearchByID() {
    System.out.println("searchByID");
    String id = "";
    Series instance = new Series();
```

```
        SeriesModel expResult = null;
        SeriesModel result = instance.searchByID(id);
        assertEquals(expResult, result);
        // TODO review the generated test code and remove the default call to fail.
        fail("The test case is a prototype.");
    }

}
/* ------------------------------------------------------------------------------------------------------------------
--------*/
/* FILE TITLE                                                                                                       */
/* ------------------------------------------------------------------------------------------------------------------
--------*/
```

Disclosure of AI Usage

Section(s) within the assessment in which generative AI was used
Assignment Question 1 and Assignment Question 2.

Name of AI tool(s) used
ChatGPT Version 1.2025.231

Purpose/intention behind use
AI was used for the following to perform the following tasks

TV Series Management Application

- Identifying and fixing logical errors, redundancies, and code readability issues (e.g. incorrect loop conditions, variable shadowing, missing return values and naming conflicts.
- Explaining Java programming concepts relevant to this project, including:
    - Access modifiers (public, private, package-private),
    - Constructors (default vs no-argument),
    - Using toString()
- Explaining and outlining JOptionPane predefined constants (JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, etc.).
- Providing assistance with HTML formatting in JOptionPane (<b>,
, <hr>).
- Assisting with the implementation of helper methods without dialogs for unit testing (i.e. addSeriesForTest(), updateSeriesForTest(), deleteSeriesForTest())
- Providing explanations of IDE warnings and error messages, such as:
    - “Cannot compare primitive to null”.
    - Variable shadowing (series vs Series).
    - Misuse of methods like .isBlank() vs .trim().isEmpty().
- Assisting with brainstorming and visualising the application flow and file structure

Referencing Console Application

- Identifying logical errors, redundancies, inconsistencies and poor code readability. For example:
    - fixing incorrect conditions and mismatched variables (referenceManager vs manager),
    - improving array bounds checks,
    - improving console output formatting
- Brainstorming the file structure
- Recommended using "~" so anonymous authors sort last in ASCII/Unicode order
- Provided regex "\\s+" for splitting names by one or more spaces.
- Clarifying Java concepts and methods, such as:
    - Abstract methods and constructors (e.g. public abstract String formatReference();)
    - Defensive copying of arrays,
    - Use of .trim() for cleaning input strings,
    - .indexOf() for finding the comma in the author string,
    - .substring() for extracting surnames,
    - .compareTo() for lexicographic string comparison in sorting,
    - scanner.nextLine() for capturing full line user input including spaces,
    - Use of continue; inside loops to skip invalid input and re-prompt,
    - System.in as the standard input stream for the Scanner
    - Use of while (true) loops for continuous prompting until valid input is entered
    - Use of .replace("..", ".") to avoid double full stops
- Guidance on user in put handling and implementing helper methods, for
- Assistance with generating Junit tests (e.g. providing test recommendations, checking test logic, etc)

Date(s) in which generative AI was used
AI was consulted from 1 September 2025 – 10 September 2025

Citations

OpenAI. 2025. Chat-GPT (Version 1.2025.231). [Large language model]. Available at: <https://chatgpt.com/> [Accessed: 10 September 2025].
