Shauna Lachelier **3/13/2025**

Unit 6 Java Generics Assignment

Catalog.java File:

```
import java.util.ArrayList;
import java.util.List;

/**
 * The generic catalog class for storing library item information.
 * Utilizing T as my Generic type parameter and extending it with the
LibraryItem class
 * so that I can extend from Library item itself or any of its subclasses.
 * @param <T>
 *
 * @author Shauna Lachelier
 */
public class Catalog<T extends LibraryItem> {
    private List<T> items; // Storing the library items in this list.

    // The constructor I'm using to instantiate a new catalog object.
    public Catalog() {
        this.items = new ArrayList<>();
    }

    /**
     * Taking the item object and storing it into the list.
     *
     * @param item
     * @throws IllegalArgumentException if a duplicate item ID exists.
     */
    public void addItem(T item) throws IllegalArgumentException {
        for (T itemExists : items) {
            if(itemExists.getItemID() == item.getItemID()) {
                throw new IllegalArgumentException("An item with the ID: "
+ item.getItemID() + " already exists. Please enter a different ID.");
            }
        }
        items.add(item);
    }
```

```
    /**
     * Providing functionality to remove an item from the list.
     *
     * @param itemID removes the item via its ID.
     */
    public void removeItem(int itemID) {
        T itemToRemove = null;
        for (T item : items) {
            if (item.getItemID() == itemID) {
                itemToRemove = item;
                break;
            }
        }
        if (itemToRemove != null) {
            items.remove(itemToRemove);
        } else {
            System.out.println("Item with ID " + itemID + " not found.");
        }
    }

    /**
     * Pulls the item object's ID and returns the item's properties and
their values.
     *
     * @param itemID
     * @return will display null if the item object with that ID doesn't
exist.
     */
    public T getItem(int itemID) {
        for (T item : items) {
            if (item.getItemID() == itemID) {
                return item;
            }
        }
        System.out.println("Item with ID " + itemID + " not found.");
        return null;
    }

    /**
     * This will display all of the items through a for loop.
     */
    public void displayCatalog() {
        for (T item : items) {
            System.out.println(item);
        }
    }
}
```

LibraryItem.java File:

```
/**
 * A class for structuring the library item objects that will be placed in
the catalog.
 *
 * @param title  Name of the item.
 * @param author Author of the item.
 * @param itemID The item's user-specified ID.
 *
 * @author Shauna Lachelier
 */
public class LibraryItem {
  private String title;
  private String author;
  private int itemID;

  /**
   * The LibraryItem constructor that will include it's title, author, and
itemID.
   *
   * @param title
   * @param author
   * @param itemID
   */
  public LibraryItem(String title, String author, int itemID) {
      this.title = title;
      this.author = author;
      this.itemID = itemID;
  }

  /**
   * This just returns the value of the title for the current item.
   * @return the item's title.
   */
  public String getTitle() {
      return title;
  }

  /**
   * returns the value of the author property for the current item.
   *
   * @return value of the author property.
   */
  public String getAuthor() {
```

```
        return author;
    }

    /**
     * returns the value of the ID for the current item.
     *
     * @return item's ID.
     */
    public int getItemID() {
        return itemID;
    }

    /**
     * Converts it all to display as a string.
     *
     * @return The string including the current item's data.
     */
    @Override
    public String toString() {
        return "LibraryItem{" +
                "title='" + title + '\'' +
                ", author='" + author + '\'' +
                ", itemID='" + itemID + '\'' +
                '}';
    }
}
```

Main.java File:

```
import java.util.Scanner;

/**
 * A CLI for users to add, remove, view and delete items in the catalog.
 *
 * @author Shauna Lachelier
 */

public class Main {
    public static void main(String[] args) {
        Catalog<LibraryItem> catalog = new Catalog<>();
        Scanner scanner = new Scanner(System.in);
        String command;

        while (true) {
```

```
            System.out.println("Enter command (add, remove, view, exit):
");
            command = scanner.nextLine();

            if (command.equalsIgnoreCase("add")) {
                System.out.println("Book or movie title: ");
                String title = scanner.nextLine();
                System.out.println("Enter author or creator: ");
                String author = scanner.nextLine();
                int itemID;

                while (true) {
                    System.out.println("Enter a unique numerical item ID:
");
                    try {
                        itemID = Integer.parseInt(scanner.nextLine());
                        LibraryItem item = new LibraryItem(title, author,
itemID);
                        catalog.addItem(item);

                        break;
                    } catch (NumberFormatException e) {
                        System.out.println("Invalid ID format. Please try
again.");
                    } catch (IllegalArgumentException e) {
                        System.out.println("That item ID already exists.
Please try again.");
                    }
                }
            } else if (command.equalsIgnoreCase("remove")) {
                System.out.println("Enter the numerical item ID for the
item you want to remove: ");
                try {
                    int itemID = Integer.parseInt(scanner.nextLine());
                    catalog.removeItem(itemID);
                } catch (NumberFormatException e) {
                    System.out.println("Invalid ID format. Please try
again.");
                }
            } else if (command.equalsIgnoreCase("view")) {
                catalog.displayCatalog();
            } else if (command.equalsIgnoreCase("exit")) {
                System.out.println("Exiting catalog...");
                break;
            } else {
                System.out.println("Invalid command. Please try again.");
            }
```

```
        }

        scanner.close();
    }
}
```

Testing.java File:

```
/**
 * A class that tests the functionality of the Catalog and LibraryItem
classes.
 * This class has methods for testing adding, removing, retrieving, and
displaying items in the catalog.
 *
 * @author Shauna Lachelier
 */

public class Testing {
    public static void main(String[] args) {
        testAddItem();
        testRemoveItem();
        testGetItem();
        testDisplayCatalog();
        testIDHasDuplicate();
    }

    /**
     * Tests the addItem method of the Catalog class
     * to ensure that an item is added and retrieved correctly.
     */
    public static void testAddItem() {
        Catalog<LibraryItem> catalog = new Catalog<>();
        LibraryItem item = new LibraryItem("Title1", "Author1", 1);
        try {
            catalog.addItem(item);
            assert item.equals(catalog.getItem(1)) : "testAddItem failed.";
            System.out.println("testAddItem succeeded.");
        } catch (IllegalArgumentException e) {
            System.out.println("testAddItem failed. Reason: " +
e.getMessage());
        }
    }

    /**
     * Tests the removeItem method of the Catalog class to
```

```
     * ensure that an item can be removed from the catalog and is not in
the list.
     */
    public static void testRemoveItem() {
        Catalog<LibraryItem> catalog = new Catalog<>();
        LibraryItem item = new LibraryItem("Title1", "Author1", 1);

        try {
            catalog.addItem(item);
            catalog.removeItem(1);
            assert catalog.getItem(1) == null : "testRemoveItem failed to
run.";
            System.out.println("testRemoveItem succeeded.");
        } catch (IllegalArgumentException e) {
            System.out.println("testRemoveItem failed. Reason: " +
e.getMessage());
        }
    }

    /**
     * Tests the getItem method of the Catalog class to
     * ensure that an item can be pulled from the catalog by its item ID.
     */
    public static void testGetItem() {
        Catalog<LibraryItem> catalog = new Catalog<>();
        LibraryItem item = new LibraryItem("Title1", "Author1", 1);
        try {
            catalog.addItem(item);
            assert item.equals(catalog.getItem(1)) : "testGetItem failed to
run.";
            System.out.println("testGetItem succeeded.");
        } catch (IllegalArgumentException e) {
            System.out.println("testGetItem failed. Reason: " +
e.getMessage());
        }
    }

    /**
     * Tests the displayCatalog method of the Catalog class to
     * ensure that all items in the catalog are displayable.
     */
    public static void testDisplayCatalog() {
        Catalog<LibraryItem> catalog = new Catalog<>();
        LibraryItem item1 = new LibraryItem("Title1", "Author1", 1);
        LibraryItem item2 = new LibraryItem("Title2", "Author2", 2);
        try{
            catalog.addItem(item1);
```

```
            catalog.addItem(item2);
            catalog.displayCatalog();
            System.out.println("testDisplayCatalog succeeded.");
        } catch (IllegalArgumentException e) {
            System.out.println("testDisplayCatalog failed. Reason: " +
e.getMessage());
        }
    }

    public static void testIDHasDuplicate() {
        Catalog<LibraryItem> catalog = new Catalog<>();
        LibraryItem item1 = new LibraryItem("Title1", "Author1", 1);
        LibraryItem item2 = new LibraryItem("Title2", "Author2", 1);
        try {
            catalog.addItem(item1);
            catalog.addItem(item2);
            System.out.println("testIDHasDuplicate has no duplicates. No
exception was thrown.");
        } catch (IllegalArgumentException e) {
            System.out.println("testIDHasDuplicate succeeded: " +
e.getMessage());
        }
    }
}
```

Screenshot: