**Exercise 1: Shuffle a List**

Shuffle the elements of an **ArrayList** using the **shuffle** method from the **Collections** class.

**Exercise 2: Find the Maximum Element in a Collection**

Find the maximum element in an **ArrayList** using the **max** method from the **Collections** class.

**Exercise 3: Reverse a List**

Reverse the elements of a **LinkedList** using the **reverse** method from the **Collections** class.

**Exercise 4: Sort a Set in Natural Order**

Sort a **TreeSet** of strings in natural order using the **sort** method from the **Collections** class.

**Exercise 5: Sort a List of User-Defined Objects**

Sort a **List** of user-defined objects (e.g., **Person** class) based on a specific attribute (e.g., age).

**Exercise 6: Find the Minimum Object in a Collection**

Find the minimum object in a **Set** of user-defined objects (e.g., **Product** class) based on a specific attribute (e.g., price).

**Exercise 7: Shuffle a List of Custom Objects**

Shuffle the elements of a **List** of user-defined objects (e.g., **Card** class representing a deck of cards).

**Exercise 8: Reverse a List of User-Defined Objects**

Reverse the order of elements in a **List** of user-defined objects (e.g., **Book** class).

**Exercise 9: Find the Maximum Object in a Collection**

Find the maximum object in a **List** of user-defined objects (e.g., **Car** class) based on a specific attribute (e.g., price).

**Exercise 10: Rotate Elements in a List of User-Defined Objects**

Rotate the elements in a **List** of user-defined objects (e.g., **Person** class) by a specified distance.

**Exercise 11: Custom Sorting of a List of User-Defined Objects**

Create a **List** of user-defined objects (e.g., **Employee** class) and implement a custom comparator to sort the list based on multiple attributes (e.g., first by department, then by salary).

**Exercise 12: Finding Median in a List of User-Defined Objects**

Create a **List** of user-defined objects (e.g., **Student** class) and find the median based on a specific attribute (e.g., exam scores).

**Exercise 13: Implement a Priority Queue with Custom Comparator**

Create a priority queue using the **PriorityQueue** class and a custom comparator to manage a list of tasks (e.g., **Task** class) based on priority.

**Exercise 14: Implement a Custom LRU Cache**

Implement a custom LRU (Least Recently Used) cache using a combination of a **LinkedHashMap** and a custom **removeEldestEntry** method in the **Cache** class.

**Exercise 15: Implement a Thread-Safe Singleton List**

Create a thread-safe singleton list using the **Collections.synchronizedList** method.

**Exercise 16: Implement a Custom Binary Search on a List of Objects**

Create a utility class (**BinarySearchUtility**) that provides a method to perform binary search on a sorted list of user-defined objects (e.g., **Person** class) based on a specified attribute (e.g., age).

**Exercise 17: Implement a Multi-Key Map using a Composite Key**

Create a multi-key map (**MultiKeyMap**) to store information about students and their courses. Each entry in the map should be identified by a composite key (e.g., a combination of student ID and course code).

**Exercise 18: Implement a Voting System with User-Defined Classes**

Create a voting system using a combination of user-defined classes (**Voter**, **Candidate**, and **Election**) and the **Collections** class to manage votes and determine the winner.

**Exercise 19: Implement a Library System with User-Defined Classes**

Create a library system using user-defined classes (**Book**, **Library**, and **Member**) and the **Collections class for sorting books:**

**Exercise 20: University Course Schedule with User-Defined Classes**

Create a university course schedule system with classes representing **Course**, **Student**, and **Schedule**. Use the **Collections** class to sort the schedule based on different criteria such as course name, student name, and class time.

**Exercise 21: Messaging System with User-Defined Classes**

Consider a messaging system with classes representing **User**, **Message**, and **MessagingSystem**. Use the **Collections** class and Java streams to implement the following operations:

1. **Sorting Messages:**
   * Implement a method in the **MessagingSystem** class to display messages sorted by sender names.
   * Implement another method to display messages sorted by receiver names.
2. **Most Recent Messages:**
   * Implement a method to display the most recent messages. Allow the user to specify the number of most recent messages to display.
3. **User-Specific Messages:**
   * Implement a method to display messages for a specific user. The method should take a **User** object as a parameter and display all messages where the user is either the sender or the receiver.

**Exercise 22: Inventory Management System with User-Defined Classes**

Consider a simplified inventory management system with classes representing **Product**, **Inventory**, and **InventoryManager**. Implement the following operations:

1. **Adding Products:**
   * Create a method in the **Inventory** class to add a product to the inventory.
2. **Updating Quantity:**
   * Create a method in the **Inventory** class to update the quantity of a specific product based on its ID.
3. **Displaying Inventory:**
   * Implement a method in the **Inventory** class to display the current inventory information, including product ID, name, price, and quantity.
4. **Sorting Products:**
   * Create a method in the **Inventory** class to sort products by their names in ascending order using the **Collections** class.
   * Implement another method to sort products by their quantities in descending order using the **Collections** class.
5. **Searching for a Product:**
   * Implement a method in the **Inventory** class to search for a product by its name. Return the product information if found; otherwise, indicate that the product is not in the inventory.
6. **Calculating Total Inventory Value:**
   * Create a method in the **Inventory** class to calculate the total value of the inventory (sum of the value of each product, where value = price \* quantity).
7. **Removing a Product:**
   * Implement a method in the **Inventory** class to remove a product from the inventory based on its ID.

Use the provided classes to structure your solution. Create sample products and test the implemented methods in the **InventoryManager** class.

**Exercise 23: Library Management System with User-Defined Classes and Collections**

Consider a library management system with classes representing **Book**, **Library**, and **LibraryManager**. Implement the following operations:

1. **Adding Books:**
   * Create a method in the **Library** class to add a book to the library.
2. **Displaying Books:**
   * Implement a method in the **Library** class to display the current library book information, including book ID, title, author, and availability status.
3. **Borrowing and Returning Books:**
   * Create methods in the **Book** class to borrow and return books. Update the availability status accordingly.
4. **Sorting Books:**
   * Implement methods in the **Library** class to sort books by title and by author using the **Collections** class.

Use the provided classes to structure your solution. Create sample books and test the implemented methods in the **LibraryManager** class.

**Exercise 24: Movie Database System with User-Defined Classes, Collections, and with Add, Show, Search, Sort, Update and Remove Options**

Consider a movie database system with classes representing **Movie**, **MovieDatabase**, and **MovieManager**. Implement the following operations:

1. **Adding Movies:**
   * Create a method in the **MovieDatabase** class to add a movie to the database.
2. **Displaying Movies:**
   * Implement a method in the **MovieDatabase** class to display the current movie information.
3. **Searching for Movies:**
   * Create methods in the **MovieDatabase** class to search for movies by director and by rating.
4. **Sorting Movies:**
   * Implement methods in the **MovieDatabase** class to sort movies by title, release year, and rating using the **Collections** class.
5. **Updating Movie Rating:**
   * Create a method in the **MovieDatabase** class to update the rating of a movie by its title.
6. **Removing Movie:**
   * Implement a method in the **MovieDatabase** class to remove a movie by its title.

Use the provided classes to structure your solution. Create sample movies and test the implemented methods in the **MovieManager** class.