**Exercise 1: HashSet**

1. Create a **HashSet** of strings and add the following elements: "apple", "banana", "cherry", "date", "fig".
2. Write code to check if the set contains "banana" and "grape" and print the results.

**Exercise 2: LinkedHashSet**

1. Create a **LinkedHashSet** of integers and add the elements 5, 2, 8, 3, 1 in this order.
2. Iterate through the set and print the elements in the order they were added.

**Exercise 3: TreeSet**

1. Create a **TreeSet** of strings and add the elements "cat", "dog", "elephant", "bat", "ant" in random order.
2. Iterate through the set and print the elements in ascending order.

**Exercise 4: HashSet Operations**

1. Create two **HashSet** objects, **set1** and **set2**, with integer elements.
2. Write code to find and print the intersection and union of these sets.

**Exercise 5: LinkedHashSet with Custom Objects**

1. Create a **LinkedHashSet** of custom objects representing books with attributes (title, author, ISBN).
2. Add a few books to the set and iterate through them, printing their details.

**Exercise 6: TreeSet with Comparator**

1. Create a **TreeSet** of strings and use a custom **Comparator** to sort the strings in descending order.
2. Add some strings to the set and iterate through them to print them in descending order.

**Exercise 7: HashSet vs. LinkedHashSet vs. TreeSet Performance**

1. Create large **HashSet**, **LinkedHashSet**, and **TreeSet** objects containing integers.
2. Measure and compare the time it takes to add, remove, and search for elements in each set.

**Exercise 8: Removing Duplicates from an ArrayList using HashSet**

1. Create an **ArrayList** of integers with duplicate values.
2. Use a **HashSet** to remove duplicates from the **ArrayList**.

**Exercise 9: Using LinkedHashSet to Maintain Insertion Order**

1. Create a **LinkedHashSet** and add a series of names in a specific order.
2. Print the names to demonstrate that the insertion order is maintained.

**Exercise 10: TreeSet and Custom Object Ordering**

1. Create a **TreeSet** of custom objects representing students with attributes (name, age).
2. Implement a custom **Comparator** to sort the students by age in ascending order.
3. Add some student objects to the set and iterate through them to print their details in ascending order of age.

**Exercise 11: HashSet vs. LinkedHashSet vs. TreeSet for Performance**

1. Create large **HashSet**, **LinkedHashSet**, and **TreeSet** objects containing integers.
2. Measure and compare the time it takes to search for a specific element in each set.

**Exercise 12: LinkedHashSet for LRU Cache**

1. Implement an LRU (Least Recently Used) cache using a **LinkedHashSet** with a maximum capacity.
2. Add methods for adding and getting items from the cache while maintaining the LRU policy.

**Exercise 13: TreeSet and Subsets**

1. Create a **TreeSet** of integers and add some random numbers.
2. Find and print a subset of numbers within a specific range, e.g., between 20 and 50.

**Exercise 14: Removing Elements from HashSet**

1. Create a **HashSet** of strings and add several elements.
2. Remove all elements starting with the letter "A".

**Exercise 15: Checking If Sets are Equal**

1. Create two **HashSet** objects, **set1** and **set2**, with integer elements.
2. Write code to check if the two sets are equal (contain the same elements).

**Exercise 16: LinkedHashSet for Maintaining Order of User Actions**

1. Create a **LinkedHashSet** to store user actions (e.g., "Login", "Browse", "Logout") in the order they occur.
2. Simulate user actions and add them to the set.
3. Print the user actions in the order they were performed.

**Exercise 17: TreeSet and Subtraction of Sets**

1. Create two **TreeSet** objects, **set1** and **set2**, with integer elements.
2. Find and print the elements that are in **set1** but not in **set2**.

**Exercise 18: LinkedHashSet for Maintaining Recent Items**

1. Create a **LinkedHashSet** to maintain a list of recently viewed items.
2. Add items in a specific order, and ensure that only the most recent items are retained (limit the capacity).

**Exercise 19: HashSet Intersection with Common Elements**

1. Create two **HashSet** objects, **set1** and **set2**, with integer elements.
2. Write code to find and print the elements that are common to both sets.

**Exercise 20: TreeSet and Higher Elements**

1. Create a **TreeSet** of integers and add some random numbers.
2. Find and print the elements that are higher (greater) than a specific number, e.g., greater than 30.

**Exercise 21: HashSet with User-Defined Class**

1. Create a **HashSet** to store a collection of user-defined objects (e.g., a **Person** class).
2. Add several **Person** objects to the set and demonstrate the ability to check for the existence of a specific person in the set.

**Exercise 22: LinkedHashSet with User-Defined Class**

1. Create a **LinkedHashSet** to store a collection of user-defined objects (e.g., a **Product** class) to maintain insertion order.
2. Add several **Product** objects to the set and iterate through them to display product details in the order they were added.

**Exercise 23: TreeSet with User-Defined Class and Custom Sorting**

1. Create a **TreeSet** to store a collection of user-defined objects (e.g., a **Book** class) and implement a custom **Comparator** to sort the books by title in alphabetical order.
2. Add several **Book** objects to the set and iterate through them to display the books in alphabetical order by title.

**Exercise 24: HashSet with User-Defined Class and Custom Equality**

1. Create a **HashSet** to store a collection of user-defined objects (e.g., a **Song** class).
2. Implement custom equality for the **Song** class by comparing the song title and artist.
3. Add several **Song** objects to the set and demonstrate the ability to check for the existence of a specific song in the set.

**Exercise 25: LinkedHashSet with User-Defined Class and Maintain Order of Events**

1. Create a **LinkedHashSet** to store a collection of user-defined objects (e.g., an **Event** class) to maintain the order of events.
2. Add several **Event** objects to the set and demonstrate that the order in which the events were added is preserved.

**Exercise 26: TreeSet with User-Defined Class and Custom Sorting**

1. Create a **TreeSet** to store a collection of user-defined objects (e.g., a **Employee** class) and implement a custom **Comparator** to sort employees by their ID in ascending order.
2. Add several **Employee** objects to the set and demonstrate that employees are sorted by their ID.

**Exercise 27: HashSet with User-Defined Class and Custom Equality**

1. Create a **HashSet** to store a collection of user-defined objects (e.g., a **Product** class).
2. Implement custom equality for the **Product** class by comparing the product name and category.
3. Add several **Product** objects to the set and demonstrate the ability to check for the existence of a specific product in the set.

**Exercise 28: LinkedHashSet with User-Defined Class and Maintain Order of Transactions**

1. Create a **LinkedHashSet** to store a collection of user-defined objects (e.g., a **Transaction** class) to maintain the order of transactions.
2. Add several **Transaction** objects to the set and demonstrate that the order in which the transactions were added is preserved.

**Exercise 29: TreeSet with User-Defined Class and Custom Sorting**

1. Create a **TreeSet** to store a collection of user-defined objects (e.g., a **Student** class) and implement a custom **Comparator** to sort students by their age in descending order.
2. Add several **Student** objects to the set and demonstrate that students are sorted by age in descending order.

**Exercise 30: HashSet with User-Defined Class**

1. Create a **HashSet** to store a collection of user-defined objects (e.g., a **Student** class).
2. Implement custom equality for the **Student** class by comparing the student's ID and name.
3. Add several **Student** objects to the set.
4. Demonstrate the ability to check for the existence of a specific student in the set.