
# Lecture-14

# Collections in Java

# Introduction to Collections

- Java Collections are objects that provide a way to store, organize, and manipulate groups of objects or data
- The Java Collections Framework is a set of classes and interfaces in Java that provides commonly reusable collection data structures and algorithms
- Collections simplify data handling by offering a consistent and flexible way to work with data structures, allowing for easier data retrieval, storage, and management

# Types of Collection Classes

- **Lists**: Lists are ordered collections that allow duplicate elements.
  - `ArrayList`
  - `LinkedList`
- **Sets**: Sets are unordered collections that do not allow duplicate elements.
  - `HashSet`
  - `TreeSet`
- **Maps**: Maps are collections that store key-value pairs.
  - `HashMap`
  - `TreeMap`

# Applications of List

- **Lists** are often used for ordered data storage, such as maintaining a list of items or managing historical records
  - **add(Object)** adds at the end of the list.
  - **remove(Object)** removes at the start of the list.
  - **list1.equals(list2)** the ordering of the elements is  taken into consideration.
  - Extra requirements to the method hashCode.
  - **list1.equals(list2**) implies that **list1.hashCode() == list2.hashCode()**

# Lists Implementation - Using ArrayList

- An **ArrayList** is a dynamic array that can grow or shrink in size, making it versatile for **lists** of items
- You can add elements to the end of the list using the add method
- Use remove to delete elements
- Iteration through the list can be done with loops, such as the for-each loop

# Coding Example - Using ArrayList

- The provided code aims to shuffle the elements passed as arguments and print the shuffled list using the **ArrayList**.

```
import java.util.*;

public class Shuffle {
    public static void main(String args[]) {
        List<String> l = new ArrayList<>();

        for (int i = 0; i < args.length; i++) {
            l.add(args[i]);
        }

        Collections.shuffle(l, new Random());
        System.out.println(l);
    }
}
```

# Applications of Set

- **Sets** are ideal for situations where you need to maintain a collection of unique elements, ensuring no duplicates are present.
  - Interface is identical.
  - Every constructor must create a collection without duplicates.
  - The operation add cannot add an element already in the set.
  - The method call set1.equals(set2) works at follows

# Set Implementation - Using HashSet

- A **HashSet** is a **set** implementation, meaning it stores unique elements without a defined order
- The add method is used to insert elements into the set
- The remove method allows for the removal of elements
- You can use contains to check if an element is present in the set

# Coding Example- Using HashSet

- The provided code aims to find and display duplicate words from the input arguments passed to the program using a **HashSet**.

```
import java.util.*;

public class FindDups {
    public static void main(String args[]) {
        Set<String> s = new HashSet<>();

        for (int i = 0; i < args.length; i++) {
            if (!s.add(args[i])) {
                System.out.println("Duplicate detected: " + args[i]);
            }
        }

        System.out.println(s.size() + " distinct words detected: " + s);
    }
}
```

# Applications of Map

- **Maps** are essential when data needs to be associated with keys, making it easy to retrieve and manipulate data efficiently. Also called an associative array or a dictionary.
  - Methods for adding and deleting
    - **put(Object key, Object value)**
    - **remove (Object key)**
  - Methods for extraction objects
    - **get (Object key)**
  - Methods to retrieve the keys, the values, and (**key, value**) pairs

# Map Implementation - Using HashMap

- A **HashMap** allows you to store and manage data in key-value pairs
- The put method adds a key-value pair to the **map**
- Use get to retrieve values based on their associated keys
- To remove a key-value pair, you can use the remove method

# Coding Example - Using HashMap

- The provided code snippet aims to count the frequency of distinct words passed as command-line arguments and then display the frequency table using a HashMap.

```
import java.util.*;

public class Freq {
    private static final Integer ONE = new Integer(1);

    public static void main(String args[]) {
        Map<String, Integer> m = new HashMap<>();

        // Initialize frequency table from command line
        for (int i = 0; i < args.length; i++) {
            Integer freq = m.get(args[i]);
            m.put(args[i], (freq == null ? ONE : new Integer(freq.intValue() + 1)));
        }

        System.out.println(m.size() + " distinct words detected:");
        System.out.println(m);
    }
}
```

# Common Operations and Methods

- **Collections** provide various methods for common operations
- add adds an element to the collection, whether it's a list, set, or map
- remove deletes elements from the collection
- get retrieves elements from a collection
- contains checks for the presence of an element within a collection
- These methods are tailored to the specific collection type and use case

# Exercises

## Unique Elements Count

Write a Java program that takes a list of integers as input and determines the count of unique elements in the list. Implement a method that receives the list and returns the count of unique integers present in it.

## Word Frequency Counter

Create a Java program that receives a sentence or multiple sentences as input strings. Break down the input into words and count the frequency of each word. Display the frequency of occurrence of each word in the input string.

# Thank You