
LETS LEARN JAVA
COLLECTIONS

Training Plan - Advanced

Topics to be covered
Collections in Java How Collections are more flexible than Arrays ArrayLists & Sets Comparing Objects Sorting Collections – Comparable, Comparator, Lambdas Maps for storing Key-Value pair Maps where Value is an object

Storing multiple data of same type

Arrays are good for storing homogeneous data. But they have a few major limitations:

1. Their size has to be declared during initialization
2. They cannot change size once declared
3. You cannot add or remove an element from middle of an array in general. The workaround is a very heavy option
4. The way to access an element is through index which could be buggy and prone to errors

To overcome such limitations, Java has come up with extensive collections framework:

1. ArrayList – similar to Array, but with loads of flexibility
2. HashSet – similar to ArrayList but does not allow duplicates
3. HashMap – Stores Objects in Key value pairs for easier search and maintenance.

Array List – Flexibility to add, remove elements

```
String[] names = { "Akash", "Charlie", "Beena", "David", "Frank", "Sameer", "Rahul" };

// How to add entries from an existing Array
List<String> nameList = new ArrayList<>();
nameList.addAll(Arrays.asList(names));

// Add one more element into ArrayList
nameList.add("Prithvi");

// remove one element
nameList.remove("Akash");
System.out.println("Contents : " + nameList + " Size : " + nameList.size());

// Add multiple new elements
nameList.addAll(Arrays.asList(names));
System.out.println("Contents : " + nameList + " Size : " + nameList.size());

// Remove element at specific position
nameList.remove(7);
System.out.println("[DUPLICATE REMOVAL] : Contents : " + nameList + " Size : " + nameList.size());
```

Array List – removing multiple items

```java
// Find and remove the duplicates from the Array
List<String> uniqueValues = new ArrayList<>();
List<String> duplicateValues = new ArrayList<>();
for (String name : nameList) {
    if (uniqueValues.contains(name)) {
        duplicateValues.add(name);
    } else {
        uniqueValues.add(name);
    }
}

System.out.println("Unique Names : " + uniqueValues);
System.out.println("Duplicate Entries : " + duplicateValues);

// removing multiple elements in one shot
List<String> filteredNames = uniqueValues;
filteredNames.removeAll(duplicateValues);
```

Array Lists – Filtering and Looping

```java
// Looping constructs
System.out.println("\n============= U S I N G    F O R   L O O P ==============\n");
List<String> namesContainingI = new ArrayList<>();

for (String name : nameList) {
    if (name.contains("i")) {
        namesContainingI.add(name);
    }
}

System.out.println(namesContainingI);

// Advanced Looping Constructs using Lambdas
System.out.println("\n============= U S I N G    F O R E A C H  ==============\n");
nameList.forEach(thisName -> {
    if (thisName.contains("i")) {
        System.out.println(thisName);
    }
});

System.out.println("\n============= U S I N G    L A M B D A   &   S T R E A M S  ==============\n");
nameList.stream().filter(name -> name.contains("i")).forEach(System.out::println);
```

Array Lists of Objects

```java
public class Dog {
    String name;
    String color;

    public Dog(String name, String color) {
        this.name = name;
        this.color = color;
    }

    @Override
    public String toString() {
        return "Dog [name=" + name + ", color=" + color + "]";
    }
}
```

```java
public static void main(String[] args) {
    List<Dog> myPets = new ArrayList<>();

    myPets.add(new Dog("Bruno", "Black"));
    myPets.add(new Dog("Tiger", "Brown"));
    myPets.add(new Dog("Spooky", "White"));
    myPets.add(new Dog("Duster", "Shaded"));
    myPets.add(new Dog("Scooby", "Brown"));

    List<Dog> myBrownPets = new ArrayList<>();

    for (Dog d : myPets) {
        if (d.color.equalsIgnoreCase("brown")) {
            myBrownPets.add(d);
        }
    }

    System.out.println(myBrownPets);

}
```

How to implement a Queue using Array List

```
public static void main(String[] args) {

    // People standing in a queue.
    // People always get added to the en
    // People always leave from front of

    push("Vikram");
    push("Rekha");
    push("Ria");
    push("Mohan");
    pop();
    pop();
    push("Ashok");
    push("Ria");
    push("Vinod");
    moveOut("Ashok");
    moveOut("Ria");
    push("Ria");
    push("Ria");

}
```

```
// Add a new person into the queue
private static void push(String newPerson) {
    peopleInQueue.add(0, newPerson);
    showQueue();
}

// Add a new person into the queue
private static void pop() {
    String personToMoveOut = peopleInQueue.get(peopleInQueue.size() - 1);
    peopleInQueue.remove(peopleInQueue.size() - 1);
    showQueue();
    System.out.println(personToMoveOut + " moved out");
}

private static void moveOut(String person) {
    peopleInQueue.remove(person);
    showQueue();
}

private static void showQueue() {
    System.out.println(peopleInQueue);
}
```

Sets – Unique values

```
public static void main(String[] args) {
    // Initializing list as part of declaration
    List<String> names = List.of("Akash", "Charlie", "Beena", "David", "Frank", "Sameer", "Rahul");

    System.out.println(names);

    // Declare an ArrayList and add duplicate entries
    List<String> namesInArrayList = new ArrayList<>();
    namesInArrayList.addAll(names);
    namesInArrayList.add("Frank");
    namesInArrayList.add("Rahul");
    namesInArrayList.add("Charlie");

    // Duplicates are allowed in array list and hence can be seen here
    System.out.println(namesInArrayList);

    // Create a Set and copy entire array into the set
    Set<String> namesInSet = new HashSet<>();
    // No duplicates
    namesInSet.addAll(namesInArrayList);

    List<String> sortedList = new ArrayList<>(namesInSet);
    Collections.sort(sortedList);

    System.out.println(sortedList);
}
```

Sets will not maintain order of insertion

Sets will also not allow duplicates

The logic of equality is handled in the .equals() method in the Object.

Comparing Objects

In order to sort objects, you need to be able to compare them. There are two ways in which you can make an object get compared with another object of same type.

1. By implementing the Comparable interface
    - Only one single default implementation for comparison
    - Used when an object can be sorted/compared in one way only
    - Done by overriding the compareTo() method
    - Collections.sort() can be used to sort

2. By writing custom comparators
    - You may create as many comparators as you need
    - Flexible implementation
    - You can create separate classes to store comparators, define inline, or even use Lambdas
    - Used where an object may be compared/sorted using different criteria at different points of time.
    - listName.sort() is used to sort, and Comparator object is passed as parameter.

Using Comparable

```
private static List<Dog> myPets = new ArrayList<>();

public static void main(String[] args) {

    myPets.add(new Dog("Bru", "Black"));
    myPets.add(new Dog("Tiger", "Brown"));
    myPets.add(new Dog("SpookySpider", "White"));
    myPets.add(new Dog("DusterBoy", "Shaded"));
    myPets.add(new Dog("Scooby", "Brown"));
    myPets.add(new Dog("Silk", "White"));
    myPets.add(new Dog("DirtyDover", "Brown"));
    myPets.add(new Dog("DarkyBlacky", "Black"));

    showPets();

    // Sort these dogs by Length of Name
    System.out.println("======= S O R T E D   U S I N G   C O M P A R A B L E ============");
    Collections.sort(myPets);
    showPets();
```

```
public class Dog implements Comparable<Dog> {
    String name;
    String color;

    public Dog(String name, String color) {
        this.name = name;
        this.color = color;
    }

    @Override
    public String toString() {
        return "Dog [name=" + name + ", color=" + color + "]";
    }

    @Override
    public int compareTo(Dog anotherDog) {
        // Default Comparison based on length of the name of the dog
        int result = this.name.length() - anotherDog.name.length();
        return result;
    }
}
```

Sorting Objects using Comparators

```
// Sort these dogs by Color
System.out.println("======= S O R T E D   U S I N G   C O M P A R A T O R ============");
Comparator<Dog> comparatorByColor = (Dog d1, Dog d2) -> {
    return d1.color.compareTo(d2.color);
};
Comparator<Dog> comparatorByName = (Dog d1, Dog d2) -> {
    return d1.name.compareTo(d2.name);
};
System.out.println("============ BY COLOR ==========");
myPets.sort(comparatorByColor);
showPets();

System.out.println("============ BY NAME ==========");
myPets.sort(comparatorByName);
showPets();
```

Sorting Objects using Lambdas

```
System.out.println("======= S O R T E D   U S I N G   L A M B D A   ============");

// Sort by Name
System.out.println("============ BY NAME ==========");
myPets.sort(Comparator.comparing(d -> d.name));
showPets();

System.out.println("============ BY COLOR REVERSED ==========");
// Sort by Color but in reversed order
myPets.sort(Comparator.comparing((Dog d) -> d.color).reversed());
showPets();

// Sort these dogs first by Color and then by Name
System.out.println("============ BY COLOR AND THEN NAME ==========");
myPets.sort(Comparator.comparing((Dog d) -> d.color).thenComparing((Dog d) -> d.name));
showPets();
```

Sorting Objects using Lambdas

```
System.out.println("======= S O R T E D   U S I N G   L A M B D A   ============");

// Sort by Name
System.out.println("============ BY NAME ==========");
myPets.sort(Comparator.comparing(d -> d.name));
showPets();

System.out.println("============ BY COLOR REVERSED ==========");
// Sort by Color but in reversed order
myPets.sort(Comparator.comparing((Dog d) -> d.color).reversed());
showPets();

// Sort these dogs first by Color and then by Name
System.out.println("============ BY COLOR AND THEN NAME ==========");
myPets.sort(Comparator.comparing((Dog d) -> d.color).thenComparing((Dog d) -> d.name));
showPets();
```

Maps (keys and values)

```
private static List<Dog> myPets = new ArrayList<>();

public static void main(String[] args) {

    myPets.add(new Dog("Bru", "Black"));
    myPets.add(new Dog("Tiger", "Brown"));
    myPets.add(new Dog("SpookySpider", "White"));
    myPets.add(new Dog("DusterBoy", "Shaded"));
    myPets.add(new Dog("Scooby", "Brown"));
    myPets.add(new Dog("Silk", "White"));
    myPets.add(new Dog("DirtyDover", "Brown"));
    myPets.add(new Dog("DarkyBlacky", "Black"));

    // Group by dogs by name, so that I can search for a pet by its name
    Map<String, Dog> myPetMap = new HashMap<>();
    for (Dog d : myPets) {
        myPetMap.put(d.name, d);
    }
```

Looping through maps

```
System.out.println("======== K E Y     V A L U E     P A I R S ============");
myPetMap.forEach((k, v) -> {
    System.out.println(k + " = { " + v + "}");
});

System.out.println("======== K E Y S ============");
for (String name : myPetMap.keySet()) {
    System.out.println(name);
}

System.out.println("======== V A L U E S ============");
for (Dog dog : myPetMap.values()) {
    System.out.println(dog);
}
```

Looping through maps

```
System.out.println("======== K E Y     V A L U E     P A I R S ============");
myPetMap.forEach((k, v) -> {
    System.out.println(k + " = { " + v + "}");
});

System.out.println("======== K E Y S ============");
for (String name : myPetMap.keySet()) {
    System.out.println(name);
}

System.out.println("======== V A L U E S ============");
for (Dog dog : myPetMap.values()) {
    System.out.println(dog);
}
```

Storing data in complex maps

```
System.out.println("========== D O G S    B Y    C O L O R ============");

// Group the pets by color, so that I can find all dogs of a specific color at once
Map<String, List<Dog>> dogsGroupedByColor = new HashMap<>();

for (Dog d : myPets) {
    // If already there is an entry for the same color, just add the dog into the existing list
    if (dogsGroupedByColor.containsKey(d.color)) {
        dogsGroupedByColor.get(d.color).add(d);

    } else {
        // If there is no entry for this color, then create a new list, add the dog into this and then
        // proceed
        List<Dog> dogs = new ArrayList<>();
        dogs.add(d);
        dogsGroupedByColor.put(d.color, dogs);
    }
}

System.out.println(dogsGroupedByColor);

// How many brown dogs in my pet shop/list
System.out.println("Number of brown dogs = " + dogsGroupedByColor.get("Brown").size());
```

Lets program the below

Take a list of employees, with multiple attributes as below. Load the Collection with about 10-15 rows of data.

```
private Long id;
private String firstName;
private String lastName;
private Boolean isMarried;
private Character gender;
private LocalDate dateOfBirth;
private LocalDate dateOfJoining;
private Integer salary;
private String region;
```

1. Find sum of salary of all unmarried people
2. Find out how many people younger than 40 are earning more than the average payout at the company
3. Create a summary of number of people, total experience and average salary by region
4. Find employees with names more than 5 characters, having an 'e' in them, married, and having less than average salary