Sets in Java

- A set keeps unique elements
- Provides methods for **adding**/**removing**/**searching** elements
- Offers very fast performance
- Types:
  - `HashSet<E>`
    - Does not guarantee the constant order of elements over time
  - `TreeSet<E>`
    - The elements are ordered incrementally
  - `LinkedHashSet<E>`
    - The order of appearance is preserved

Methods

- Initialization:

```
Set<String> hash = new HashSet<String>();
```

- For easy reading you can use diamond inference syntax:

```
Set<String> tree = new TreeSet<>();
```

- `.size()`
- `.isEmpty()`

```
Set<String> hash = new HashSet<>();
System.out.println(hash.size());      // 0
System.out.println(hash.isEmpty());   // True
```

Methods

- Initialization

```
Map<String, Integer> hash = new HashMap<String, Integer>();
```

Type of keys

Type of values

- .size()
- .isEmpty()

```
Map<String, Integer> hash = new HashMap<>();
System.out.println(hash.size());     // 0
System.out.println(hash.isEmpty());  // True
```

HashMap<K, V>, TreeMap<K, V>, LinkedHashMap<K, V>

- `size()` - the number of key-value pairs
- `keySet()` - a set of unique keys
- `values()` - a collection of all values
- Basic operations - `put()`, `remove()`, `clear()`
- Boolean methods:
  - `containsKey()` - checks if a key is present in the Map
  - `containsValue()` - checks if a value is present in the Map

Sorting Collections

- Using **sorted()** to sort collections:

Ascending (Natural) Order

```
nums = nums.stream()
             .sorted((n1, n2) -> n1.compareTo(n2))
             .collect(Collectors.toList());
```

Descending Order

```
nums = nums.stream()
             .sorted((n1, n2) -> n2.compareTo(n1))
             .collect(Collectors.toList());
```

Sorting Collections by Multiple Criteria

- Using **sorted()** to sort collections by multiple criteria:

```
Map<Integer, String> products = new HashMap<>();
products.entrySet()
     .stream()
     .sorted((e1, e2) -> {
        int res = e2.getValue().compareTo(e1.getValue());
        if (res == 0)
          res = e1.getKey().compareTo(e2.getKey());
        return res; })
     .forEach(e -> System.out.println(e.getKey() + " " + e.getValue()));
```

Second criteria

Terminates the stream

Sorting in Ascending Order by Value

```
Map<String, Integer> mp = new HashMap<>();
        mp.put("Aries", 1);
        mp.put("Taurus", 2);
        mp.put("Gemini", 3);
Map<String, Integer> resultMap = mp.entrySet()
                .stream()
                .sorted(Map.Entry.<String, Integer>comparingByValue())
                .collect(Collectors.toMap(Map.Entry::getKey,
                                          Map.Entry::getValue,(e1, e2) -> e1, Linked
HashMap::new));
```

Using Functional ForEach (1)

```
Map<String, ArrayList<Integer>> arr = new HashMap<>();
arr.entrySet().stream()
    .sorted((a, b) -> {
      if (a.getKey().compareTo(b.getKey()) == 0) {
        int sumFirst = a.getValue().stream().mapToInt(x -> x).sum();
        int sumSecond = b.getValue().stream().mapToInt(x -> x).sum();
        return sumFirst - sumSecond;
      }
      return b.getKey().compareTo(a.getKey());
    })
```

Second criteria

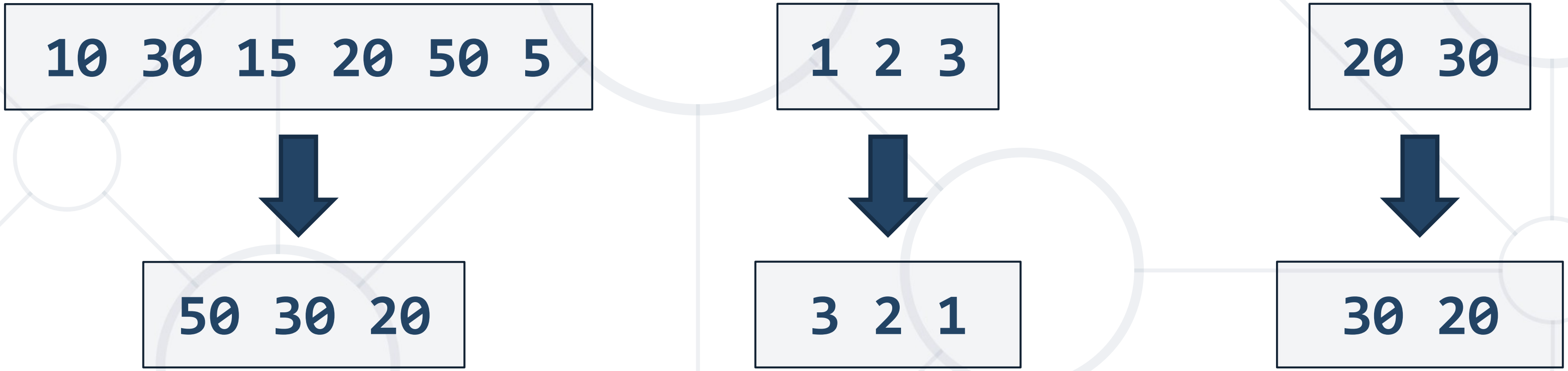
Descending sorting

Using Functional ForEach (2)

```
.forEach(pair -> {
  System.out.println("Key: " + pair.getKey());
  System.out.print("Value: ");
  pair.getValue().sort((a, b) -> a.compareTo(b));
  for (int num : pair.getValue()) {
    System.out.printf("%d ", num);
  }
  System.out.println();
});
```

Problem: Largest 3 Numbers

- Read a list of numbers
- Print the largest 3, if there are less than 3, print all of them



```
10 30 15 20 50 5
```

```
50 30 20
```

```
1 2 3
```

```
3 2 1
```

```
20 30
```

```
30 20
```

Check your solution here: https://judge.softuni.org/Contests/1462/

Solution: Largest 3 Numbers

```
List<Integer> nums = Arrays
                .stream(sc.nextLine().split(" "))
                .map(e -> Integer.parseInt(e))
                .sorted((n1, n2) -> n2.compareTo(n1))
                .limit(3)
                .collect(Collectors.toList());
for (int num : nums) {
            System.out.print(num + " ");
        }
```

Check your solution here: https://judge.softuni.org/Contests/1462/