Java Collections

Gerardo Andres Almaguer Ramos

What are Collections in Java?

- Collections are data structures that group objects.

- They allow efficient storage, manipulation, and access to data.

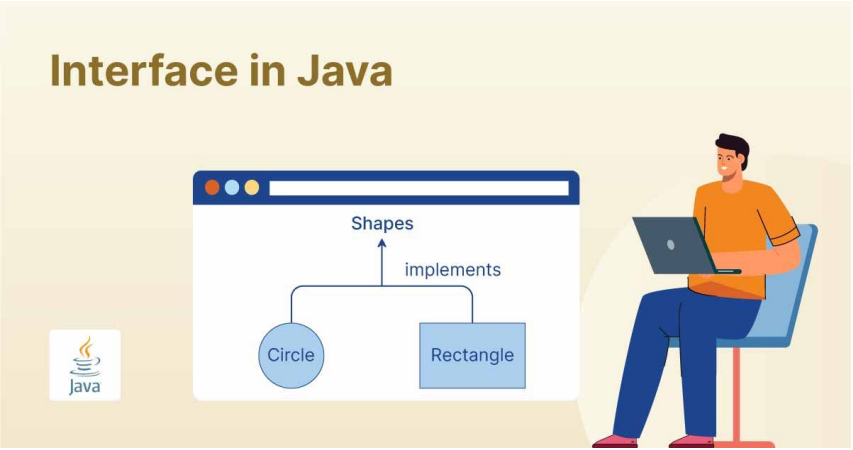
Key Collection Interfaces:

- Collection
- List
- Set
- Queue
- Map (though not extending Collection, it is crucial for data management)

Main Interfaces

- **Collection**: The root of the collection hierarchy.
  - Subinterfaces: **List, Set, Queue**

- **Map**: Stores key-value pairs.
  - Does not extend **Collection**



The `List` Interface

- Stores elements in a specific order.
- Allows duplicate elements.
- **Common Implementations:**
  - ArrayList
  - LinkedList

Code Example:

```java
import java.util.ArrayList;
import java.util.List;

public class ListExample {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");
        list.add("JavaScript");
        System.out.println(list);
    }
}
```

The `Set` Interface

- Stores elements with no duplicates.
- Does not guarantee a specific order.
- **Common Implementations:**
  - HashSet
  - LinkedHashSet
  - TreeSet

Code Example:

```java
import java.util.HashSet;
import java.util.Set;

public class SetExample {
    public static void main(String[] args) {
        Set<String> set = new HashSet<>();
        set.add("Java");
        set.add("Python");
        set.add("Java"); // Duplicate, will not be added
        System.out.println(set);
    }
}
```

The Queue Interface

- Allows storing elements in a sequence.
- Ideal for managing tasks in FIFO (First In, First Out) order.
- **Common Implementations:**
  - LinkedList
  - PriorityQueue

Code Example:

```java
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<String> queue = new LinkedList<>();
        queue.add("Task1");
        queue.add("Task2");
        queue.add("Task3");
        System.out.println(queue);
        System.out.println("Processing: " + queue.poll());
        System.out.println(queue);
    }
}
```

The `Map` Interface

- Stores key-value pairs.
- Keys are unique; values can be duplicated.
- **Common Implementations:**
  - HashMap
  - LinkedHashMap
  - TreeMap

Code Example:

```
import java.util.HashMap;
import java.util.Map;

public class MapExample {
    public static void main(String[] args) {
        Map<String, String> map = new HashMap<>();
        map.put("Java", "A high-level programming language.");
        map.put("Python", "A versatile scripting language.");
        map.put("JavaScript", "A language for web development.");
        System.out.println(map);
    }
}
```

Comparing Implementations

- **ArrayList vs. LinkedList:** Fast access vs. Fast insertions.
- **HashSet vs. TreeSet:** Unordered vs. Naturally ordered.
- **HashMap vs. TreeMap:** Unordered vs. Ordered by keys.

Conclusion

In summary, Java collections offer a variety of powerful tools for managing data efficiently. By understanding the core interfaces—List, Set, Queue, and Map—along with their common implementations, you can select the most suitable data structure for your needs. Whether you're dealing with ordered lists, unique sets, FIFO queues, or key-value pairs, choosing the right collection can significantly impact performance and ease of use. Mastery of these collections not only enhances code efficiency but also improves overall software design.