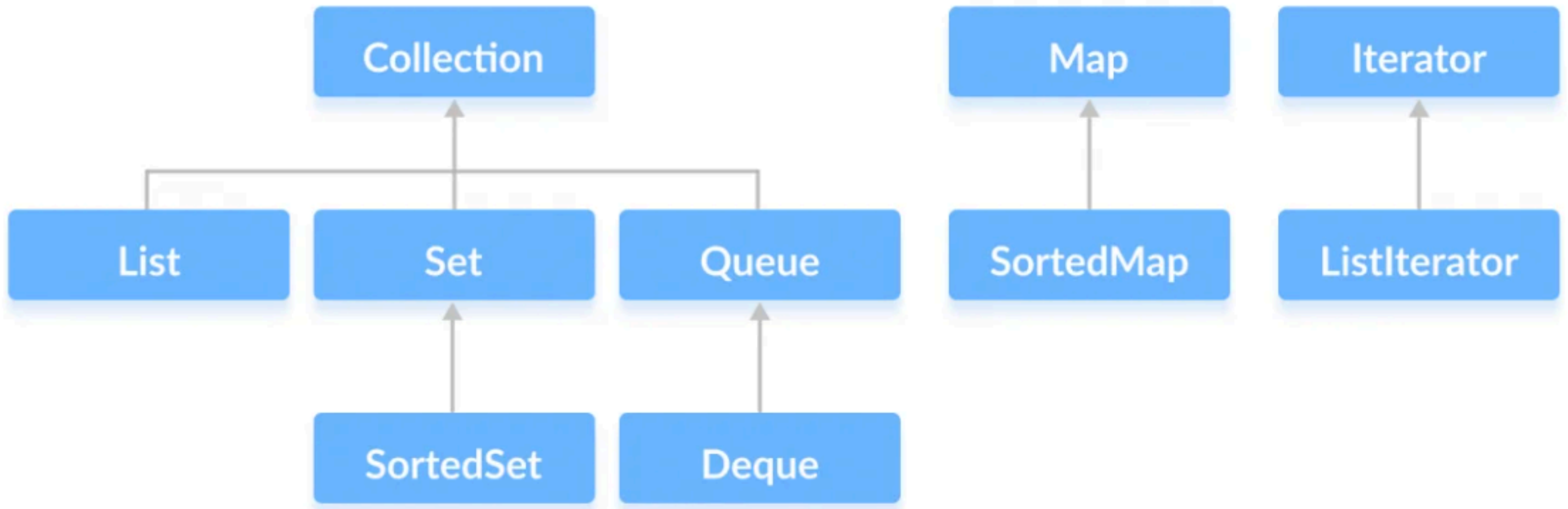
Java Course

Collections

Java Collections Framework

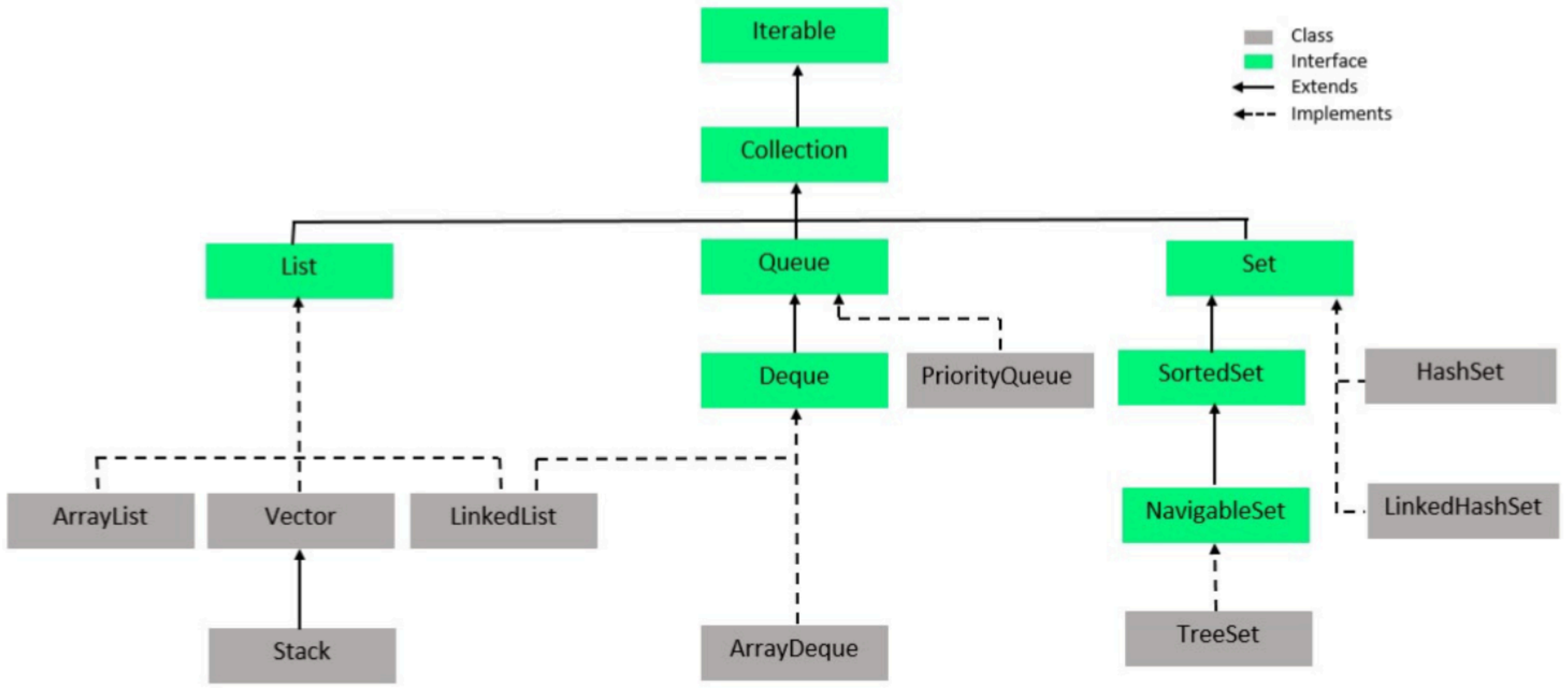
- Java Collections Framework provides a set of interfaces and classes to implement various data structures and algorithms
- These interfaces include several methods to perform different operations on collections
- Advantage of using Collections Framework:
    - we do not have to implement these data structures and algorithms manually
    - efficiency - highly optimized

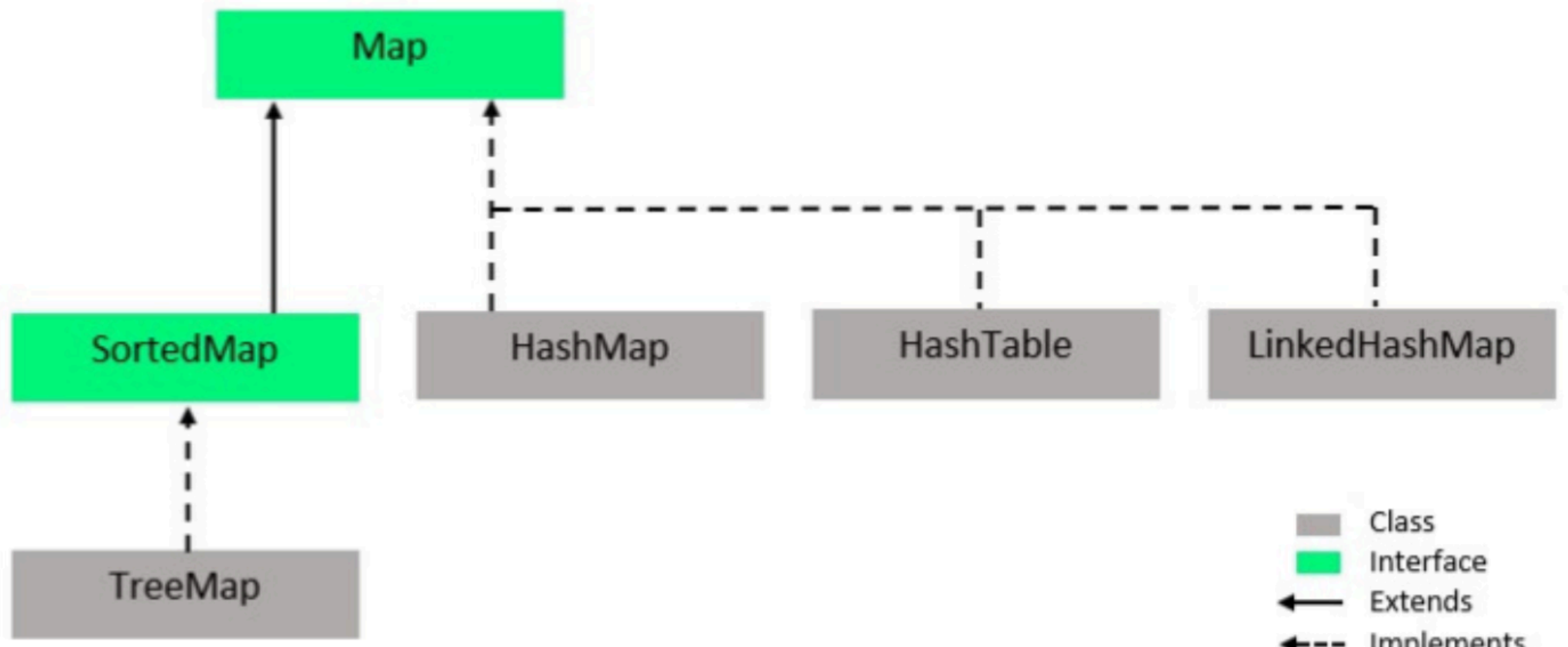
Java Collections Framework



Java `Collection` interface

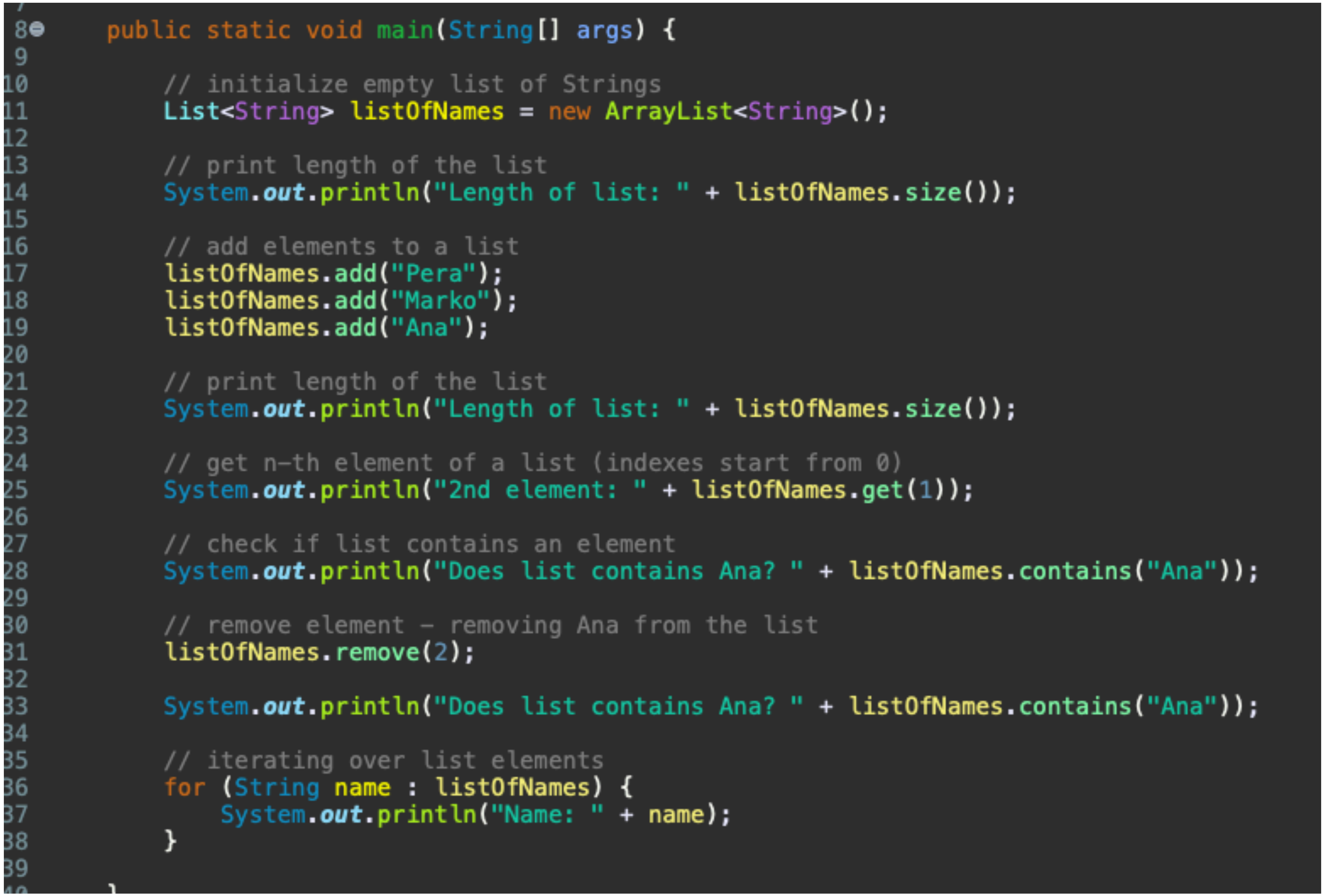
- The `Collection` interface is the root interface of the Collections framework hierarchy
- Java does not provide direct implementation of the Collection interface but provides implementations of its subinterfaces like `List`, `Set`, `Queue`
- Java Collections Framework includes other intefaces as well: `Map` and `Iterator`


Iterable
Collection
List
Queue
Set
Deque
PriorityQueue
SortedSet
HashSet
ArrayList
Vector
LinkedList
NavigableSet
LinkedHashSet
Stack
ArrayDeque
TreeSet
Class
Interface
Extends
Implements


Map
SortedMap
HashMap
HashTable
LinkedHashMap
TreeMap
Class
Interface
Extends
Implements

List

- List is an ordered collection of objects, where same object may appear more than once
- Methods:
    - `add()` - adds an element to a list
    - `get()` - accessing elements
    - `remove()` - removes an element from the list
    - `size()` - returns length of a list
    - `contains()` - check if the list contains a certain element
- Implementations:
    - `ArrayList`, `LinkedList`, `Vector`,..

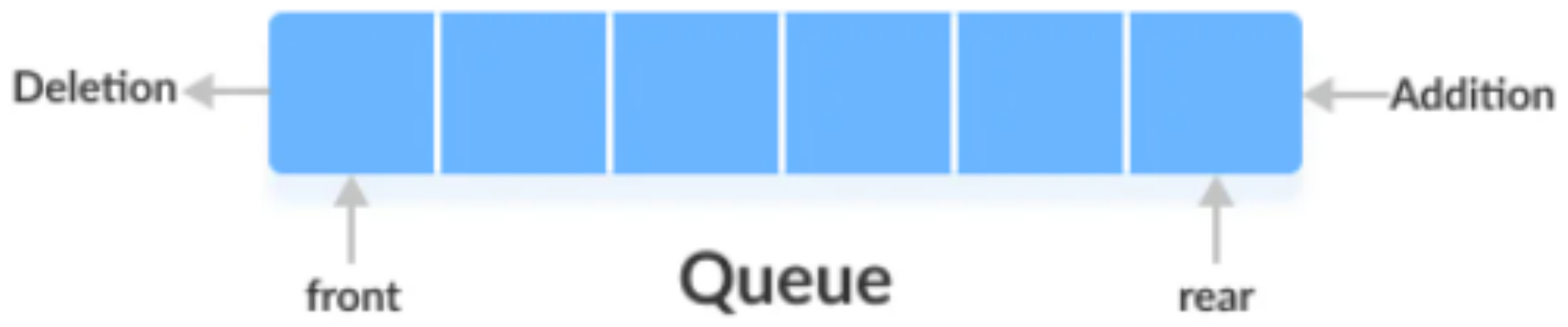


```
    public static void main(String[] args) {

        // initialize empty list of Strings
        List<String> listOfNames = new ArrayList<String>();

        // print length of the list
        System.out.println("Length of list: " + listOfNames.size());

        // add elements to a list
        listOfNames.add("Pera");
        listOfNames.add("Marko");
        listOfNames.add("Ana");

        // print length of the list
        System.out.println("Length of list: " + listOfNames.size());

        // get n-th element of a list (indexes start from 0)
        System.out.println("2nd element: " + listOfNames.get(1));

        // check if list contains an element
        System.out.println("Does list contains Ana? " + listOfNames.contains("Ana"));

        // remove element - removing Ana from the list
        listOfNames.remove(2);

        System.out.println("Does list contains Ana? " + listOfNames.contains("Ana"));

        // iterating over list elements
        for (String name : listOfNames) {
            System.out.println("Name: " + name);
        }

    }
```

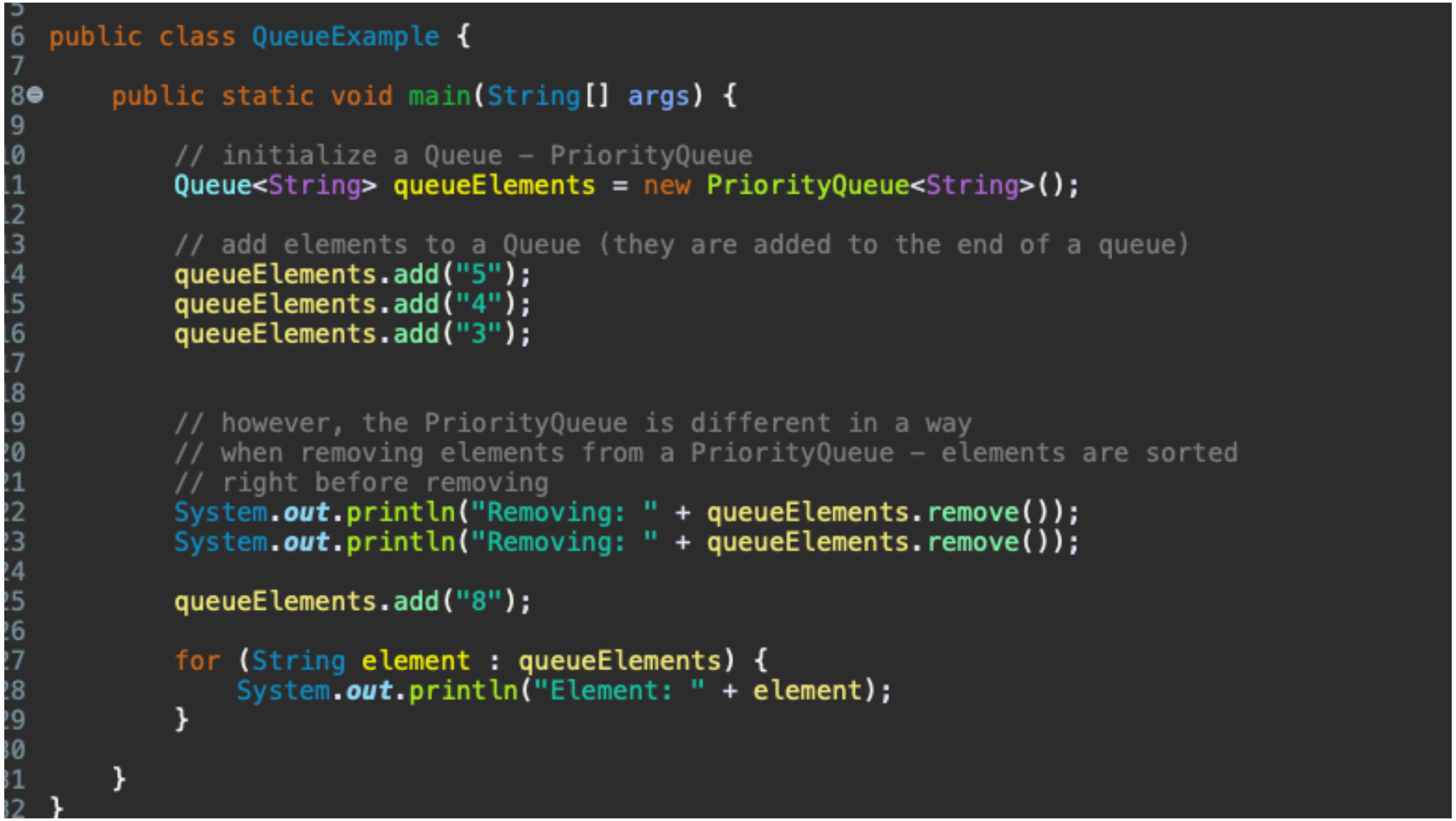
ArrayList Exercise

- Write a Java program to create a new array list, add some colours (String) and print out the collection
- Write a Java program to iterate through all elements in a array list
- Write a Java program to insert an element into the array list at the last position
- Write a Java program to insert an element into the array list at the first position
- Write a Java program to update specific array element by given element
- Write a Java program to retrieve an element at a specified index from a given array list
- Write a Java program to search for an element in array list

Queue



- A queue is an ordered collection of elements with a functionality of a queue (FIFO - **F**irst **I**n **F**irst **O**ut)
- Methods:
  - `add()` - insert element into the queue (add to the end of a queue)
  - `remove()` - remove element from the queue (remove from the front of the queue)
  - `element()` - returns head element of the queue
- Implementations:
  - `PriorityQueue`, `ArrayDequeue`,...



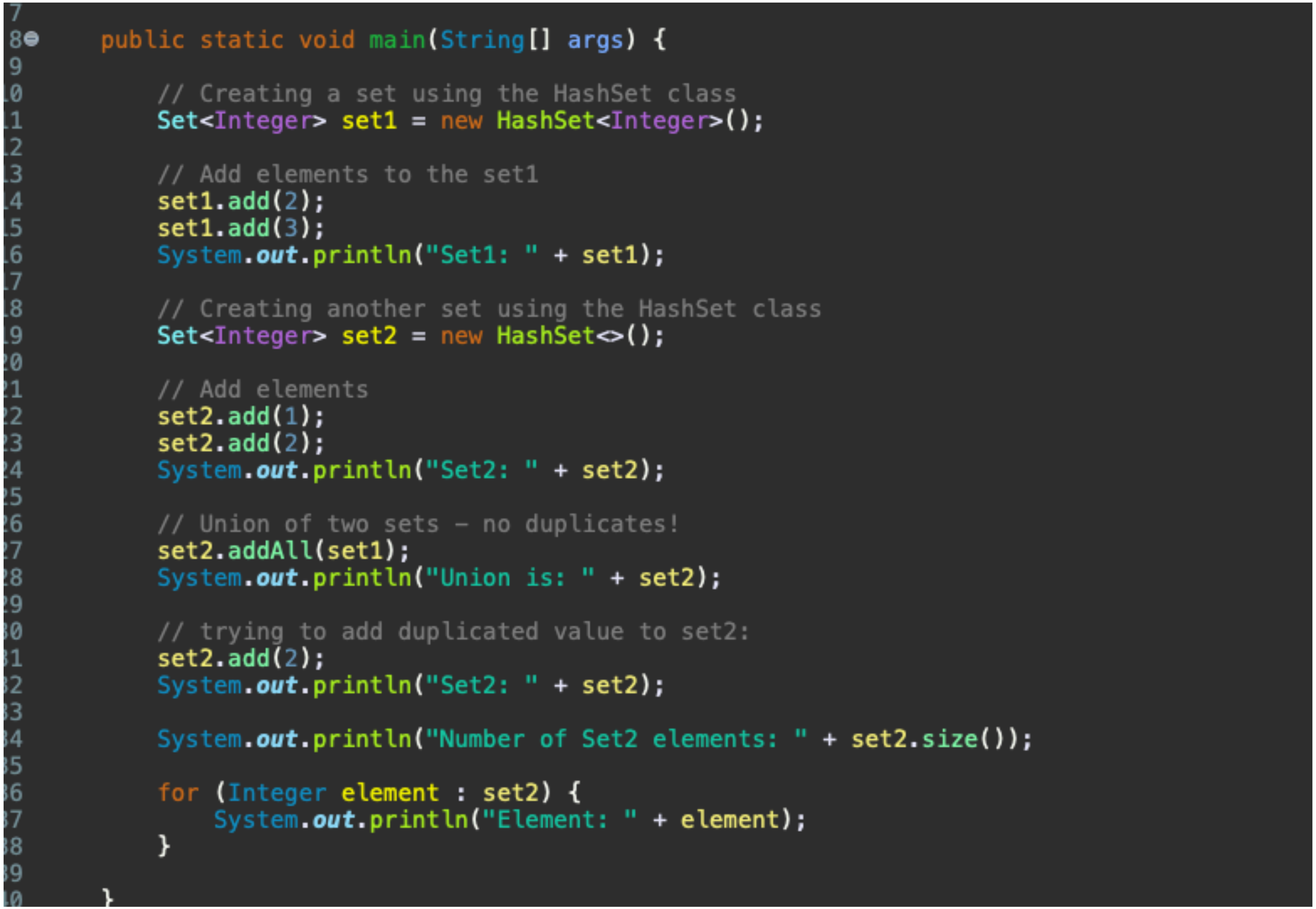
```java
public class QueueExample {

    public static void main(String[] args) {

        // initialize a Queue - PriorityQueue
        Queue<String> queueElements = new PriorityQueue<String>();

        // add elements to a Queue (they are added to the end of a queue)
        queueElements.add("5");
        queueElements.add("4");
        queueElements.add("3");


        // however, the PriorityQueue is different in a way
        // when removing elements from a PriorityQueue - elements are sorted
        // right before removing
        System.out.println("Removing: " + queueElements.remove());
        System.out.println("Removing: " + queueElements.remove());

        queueElements.add("8");

        for (String element : queueElements) {
            System.out.println("Element: " + element);
        }

    }
}
```

PriorityQueue Exercise

- Write a Java program to create a new priority queue, add some colours (String) and print out the elements of the priority queue
- Write a Java program to iterate through all elements in priority queue
- Write a Java program to retrieve the first element of the priority queue
- Write a Java program to retrieve and remove the first element

Set

- A set is unordered collection of objects, and it cannot contains duplicates
- It acts like a mathematical set
- Methods:
  - `add()` - adds an element to a set
  - `remove()` - removes an element from a set
  - `size()` - returns number of set elements
  - `contains()` - check if the set contains a certain element
  - `clear()` - removes all the elements from the set
- Implementations:
  - `HashSet`, `LinkedHashSet`, `TreeSet`



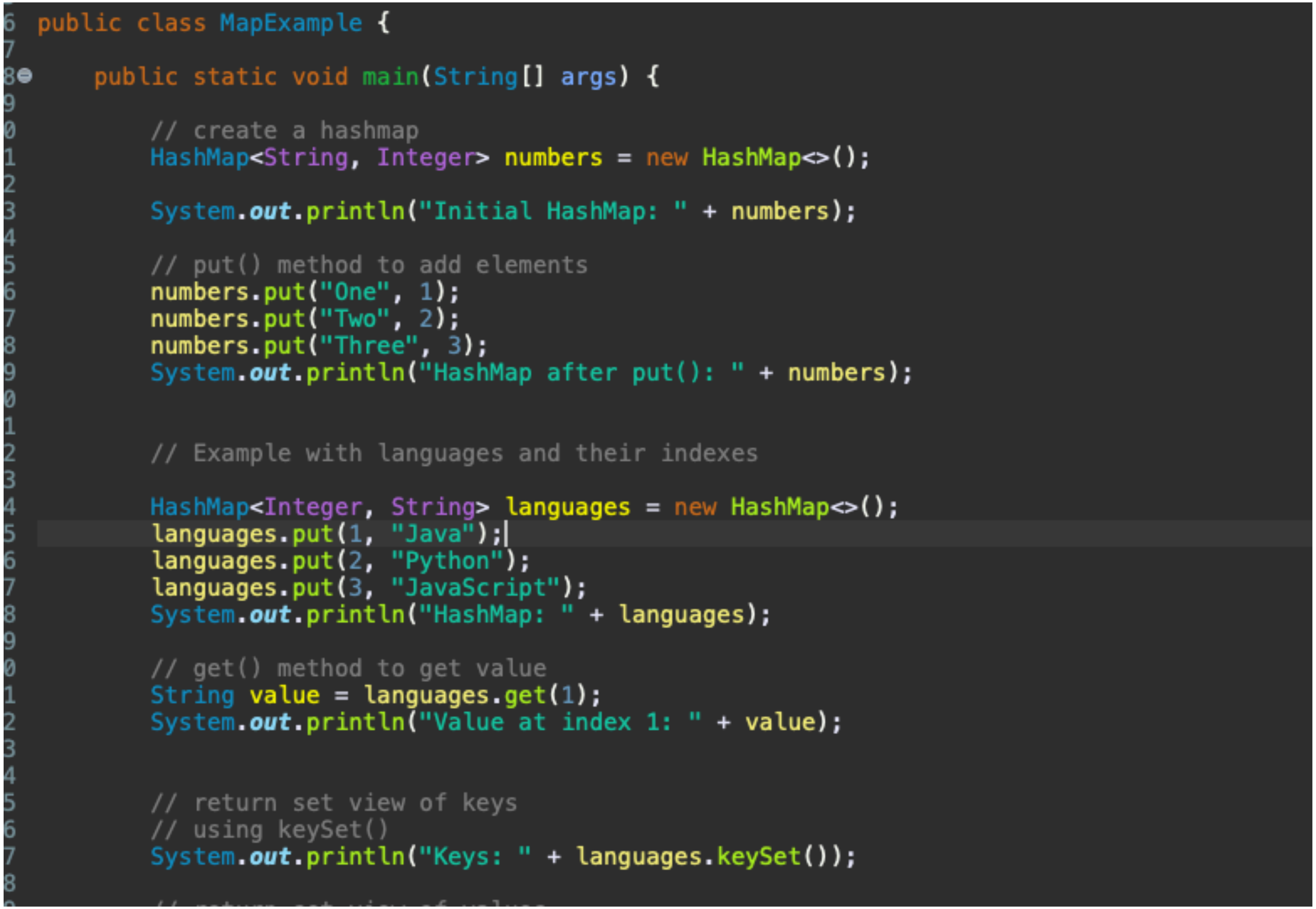
```java
    public static void main(String[] args) {

        // Creating a set using the HashSet class
        Set<Integer> set1 = new HashSet<Integer>();

        // Add elements to the set1
        set1.add(2);
        set1.add(3);
        System.out.println("Set1: " + set1);

        // Creating another set using the HashSet class
        Set<Integer> set2 = new HashSet<>();

        // Add elements
        set2.add(1);
        set2.add(2);
        System.out.println("Set2: " + set2);

        // Union of two sets - no duplicates!
        set2.addAll(set1);
        System.out.println("Union is: " + set2);

        // trying to add duplicated value to set2:
        set2.add(2);
        System.out.println("Set2: " + set2);

        System.out.println("Number of Set2 elements: " + set2.size());

        for (Integer element : set2) {
            System.out.println("Element: " + element);
        }

    }
```

HashSet Exercise

- Write a Java program to create a new hash set and add some numbers to it
- Write a Java program to iterate through all elements in a hash set
- Write a Java program to get the number of elements in a hash set
- Write a Java program to empty an hash set
- Write a Java program to simulate union on two hash sets
    - [1,2] UNION [2,3] = [1, 2, 3]
- Write a Java program to simulate intersection of two hash sets
    - [1, 2] INTERSECT [2,3] = [2]

Map

- Map interface provides functionality of the map data structure
- Elements of Map are stored in a key/value pairs
- Keys are unique - no duplicate keys
- Each key is associated with a single value



```java
public class MapExample {

    public static void main(String[] args) {

        // create a hashmap
        HashMap<String, Integer> numbers = new HashMap<>();

        System.out.println("Initial HashMap: " + numbers);

        // put() method to add elements
        numbers.put("One", 1);
        numbers.put("Two", 2);
        numbers.put("Three", 3);
        System.out.println("HashMap after put(): " + numbers);


        // Example with languages and their indexes

        HashMap<Integer, String> languages = new HashMap<>();
        languages.put(1, "Java");
        languages.put(2, "Python");
        languages.put(3, "JavaScript");
        System.out.println("HashMap: " + languages);

        // get() method to get value
        String value = languages.get(1);
        System.out.println("Value at index 1: " + value);


        // return set view of keys
        // using keySet()
        System.out.println("Keys: " + languages.keySet());

```