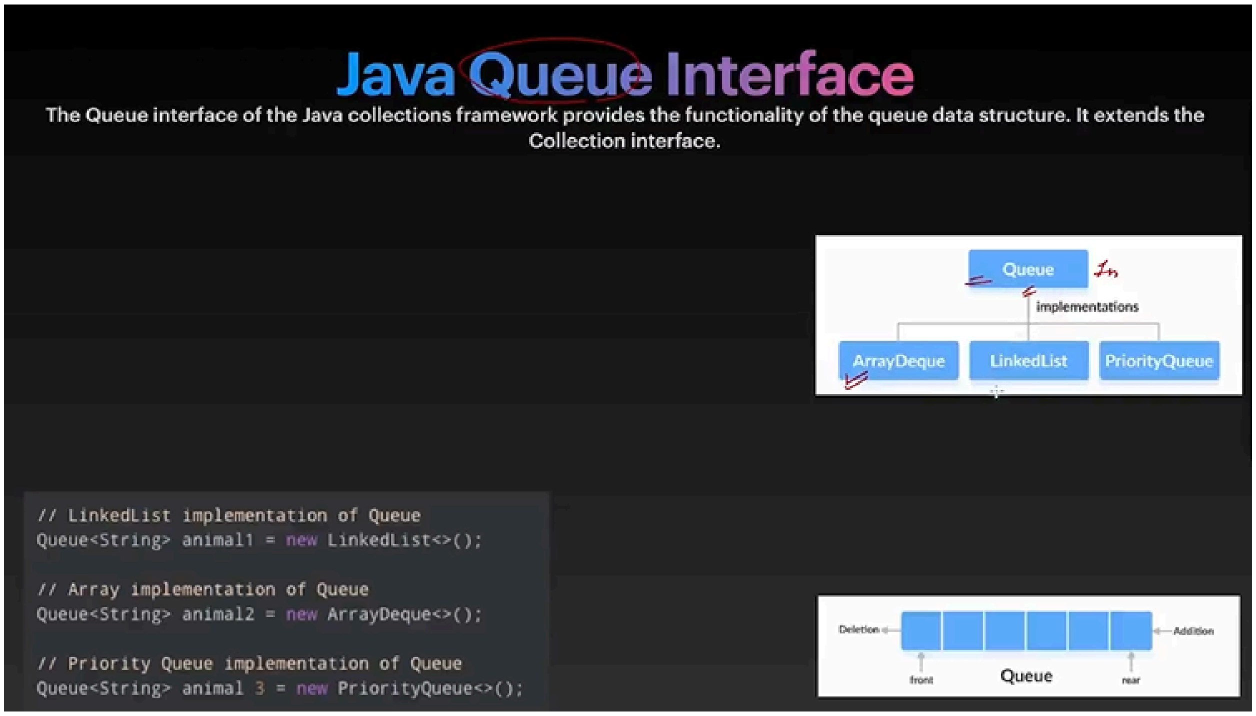
Java Collection Framework-2

INSTRUCTOR: Love Babbar

Module 2

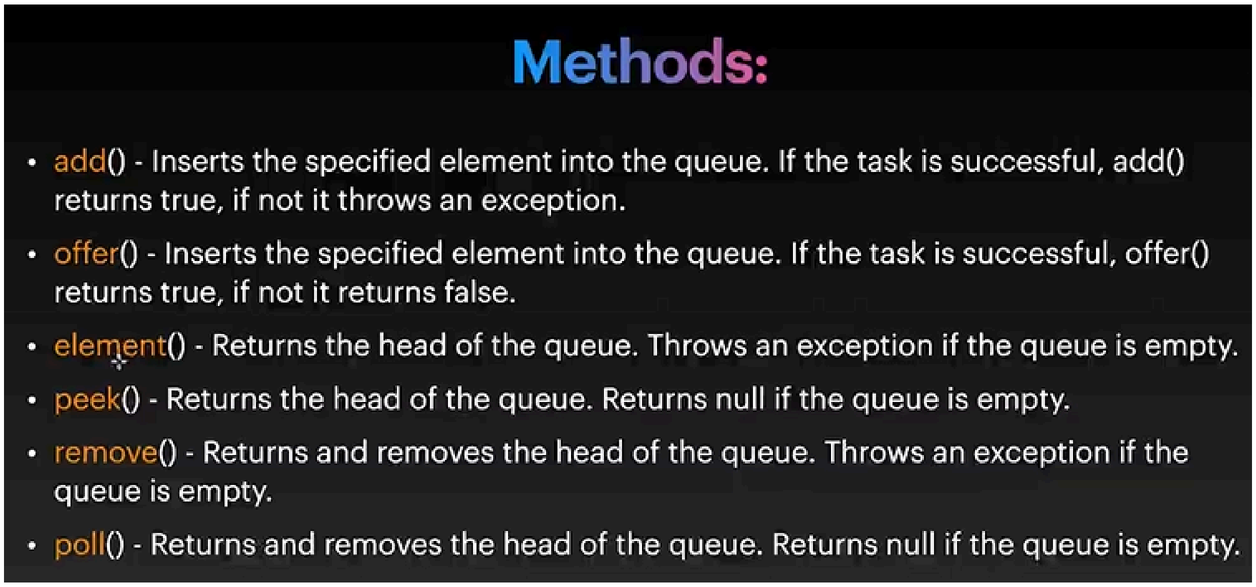
27 November 2024



Java Queue Interface

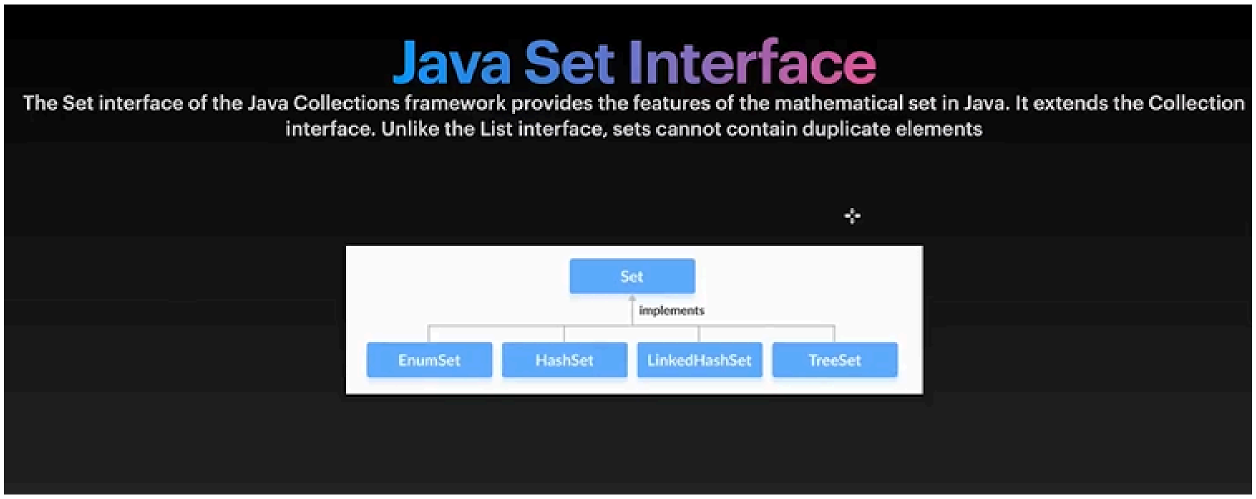
The Queue interface of the Java collections framework provides the functionality of the queue data structure. It extends the Collection interface.

```
// LinkedList implementation of Queue
Queue<String> animal1 = new LinkedList<>();

// Array implementation of Queue
Queue<String> animal2 = new ArrayDeque<>();

// Priority Queue implementation of Queue
Queue<String> animal 3 = new PriorityQueue<>();
```



Methods:

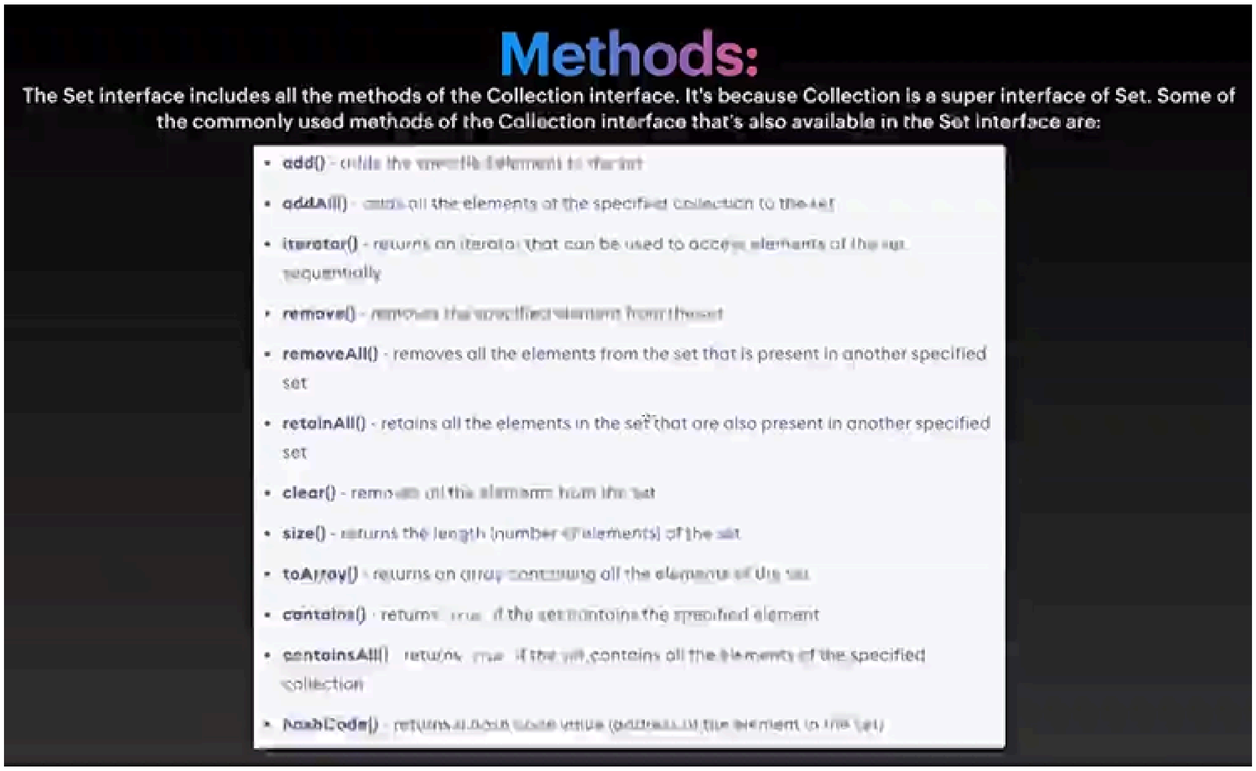
- add() - Inserts the specified element into the queue. If the task is successful, add() returns true, if not it throws an exception.
- offer() - Inserts the specified element into the queue. If the task is successful, offer() returns true, if not it returns false.
- element() - Returns the head of the queue. Throws an exception if the queue is empty.
- peek() - Returns the head of the queue. Returns null if the queue is empty.
- remove() - Returns and removes the head of the queue. Throws an exception if the queue is empty.
- poll() - Returns and removes the head of the queue. Returns null if the queue is empty.



Java Set Interface

The Set interface of the Java Collections framework provides the features of the mathematical set in Java. It extends the Collection interface. Unlike the List interface, sets cannot contain duplicate elements

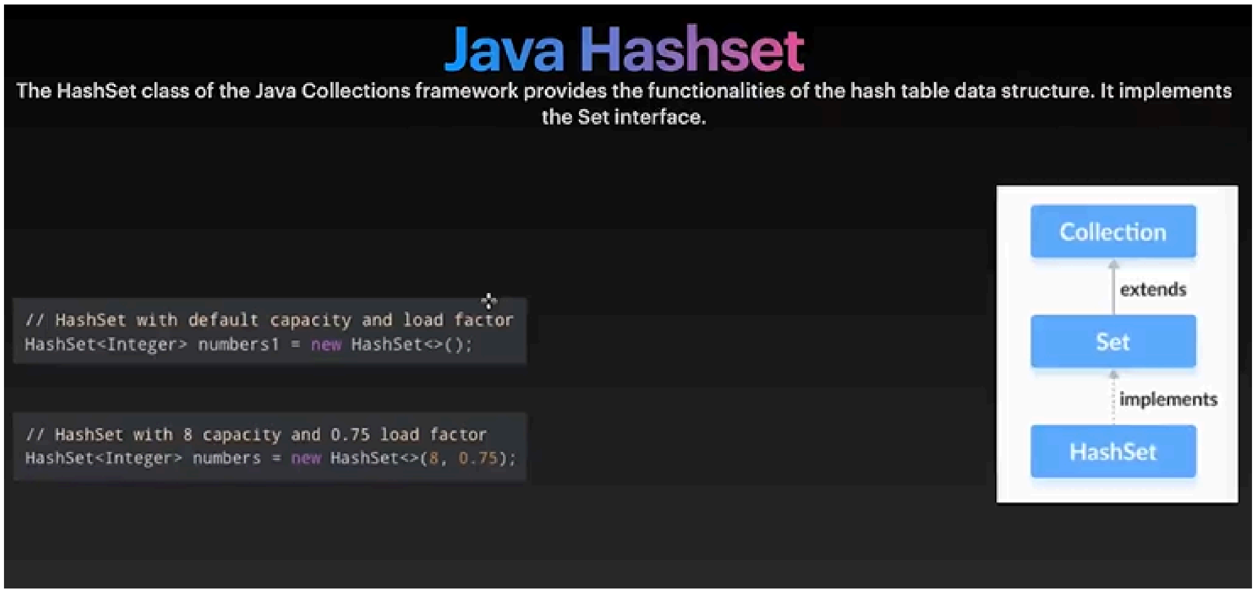
Stores unique elements



Methods:

The Set interface includes all the methods of the Collection interface. It's because Collection is a super interface of Set. Some of the commonly used methods of the Collection interface that's also available in the Set interface are:

- **add()** - adds the specified element to the set
- **addAll()** - adds all the elements of the specified collection to the set
- **iterator()** - returns an iterator that can be used to access elements of the set sequentially
- **remove()** - removes the specified element from the set
- **removeAll()** - removes all the elements from the set that is present in another specified set
- **retainAll()** - retains all the elements in the set that are also present in another specified set
- **clear()** - removes all the elements from the set
- **size()** - returns the length (number of elements) of the set
- **toArray()** - returns an array containing all the elements of the set
- **contains()** - returns true if the set contains the specified element
- **containsAll()** - returns true if the set contains all the elements of the specified collection
- **hashCode()** - returns a hash code value (address of the element in the set)

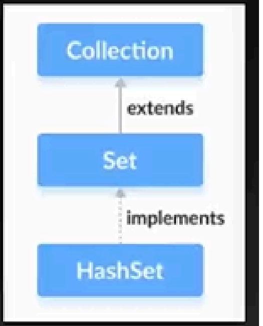


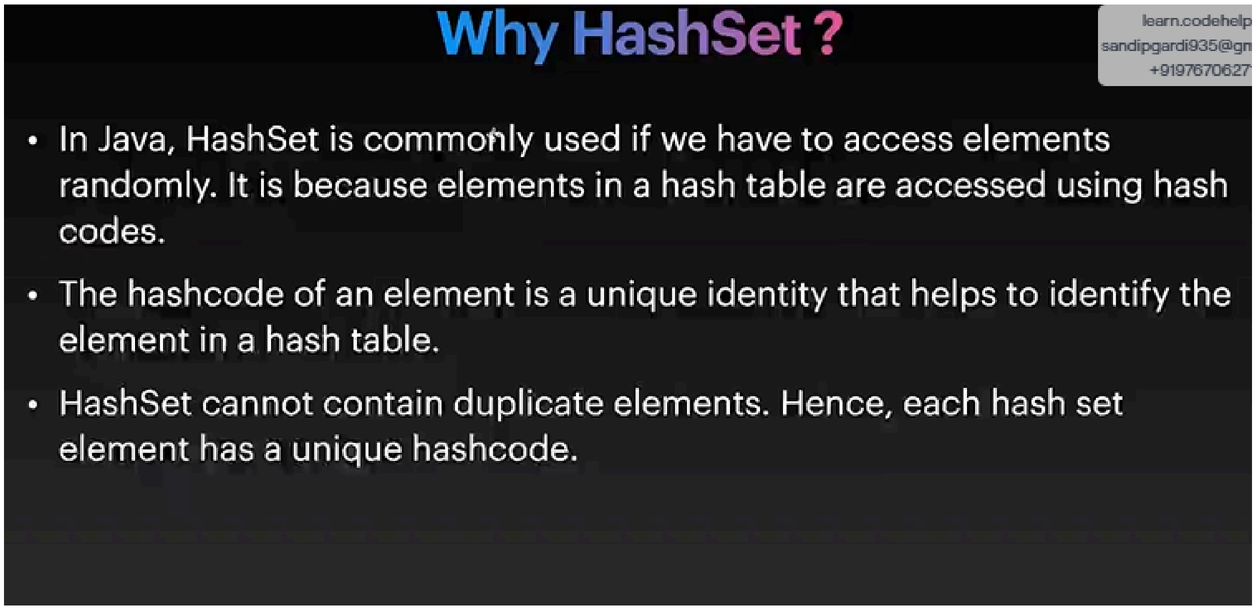
Java Hashset

The HashSet class of the Java Collections framework provides the functionalities of the hash table data structure. It implements the Set interface.

```
// HashSet with default capacity and load factor
HashSet<Integer> numbers1 = new HashSet<>();
```

```
// HashSet with 8 capacity and 0.75 load factor
HashSet<Integer> numbers = new HashSet<>(8, 0.75);
```



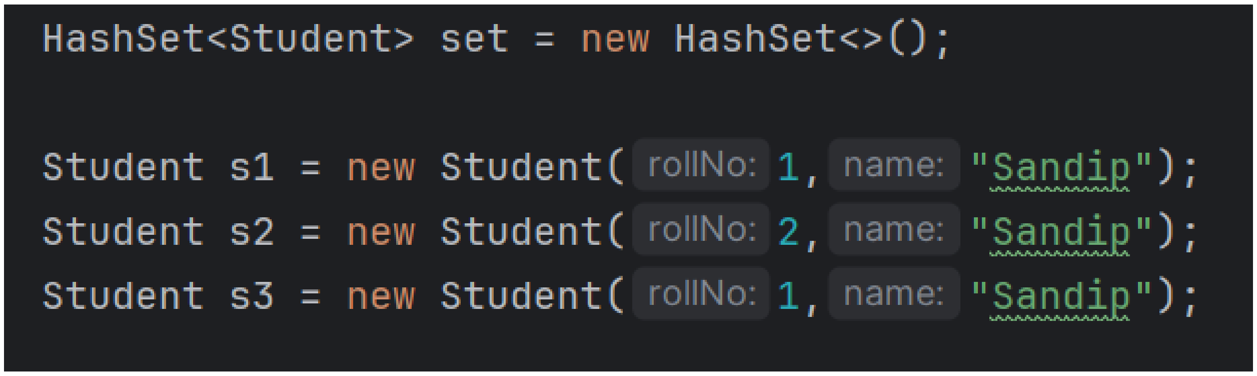


Why HashSet ?

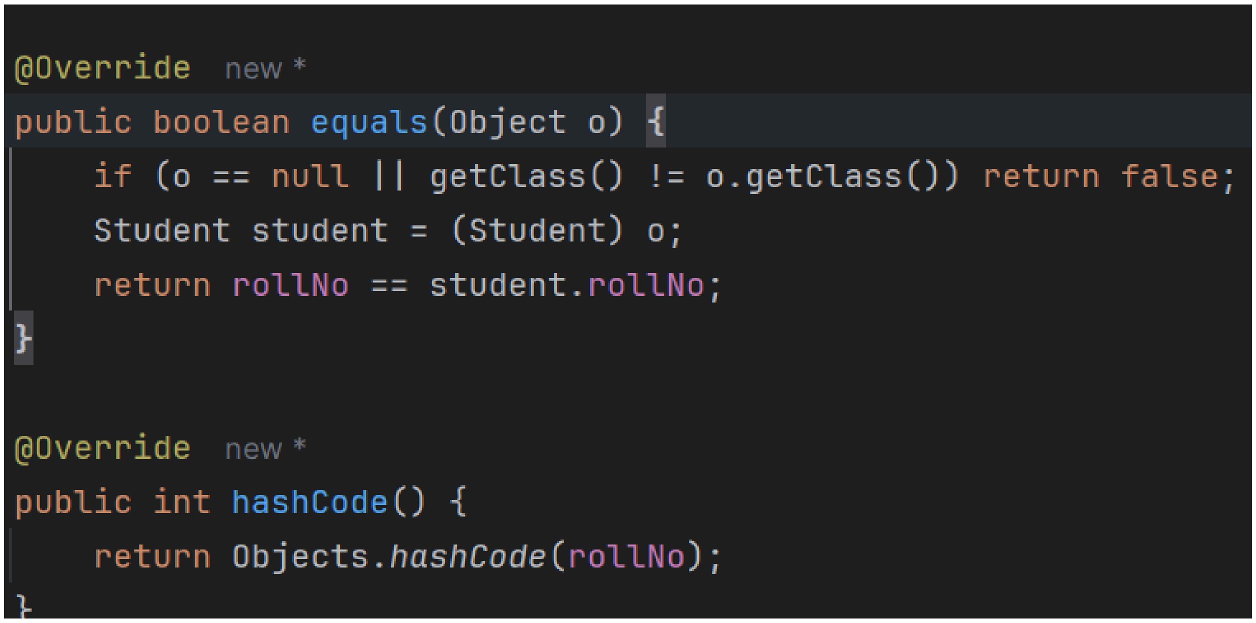
- In Java, HashSet is commonly used if we have to access elements randomly. It is because elements in a hash table are accessed using hash codes.
- The hashcode of an element is a unique identity that helps to identify the element in a hash table.
- HashSet cannot contain duplicate elements. Hence, each hash set element has a unique hashcode.

Custom object

in custom object you need to override equals and hashcode function



```
HashSet<Student> set = new HashSet<>();

Student s1 = new Student( rollNo: 1, name: "Sandip");
Student s2 = new Student( rollNo: 2, name: "Sandip");
Student s3 = new Student( rollNo: 1, name: "Sandip");
```



```
@Override  new *
public boolean equals(Object o) {
    if (o == null || getClass() != o.getClass()) return false;
    Student student = (Student) o;
    return rollNo == student.rollNo;
}

@Override  new *
public int hashCode() {
    return Objects.hashCode(rollNo);
}
```