

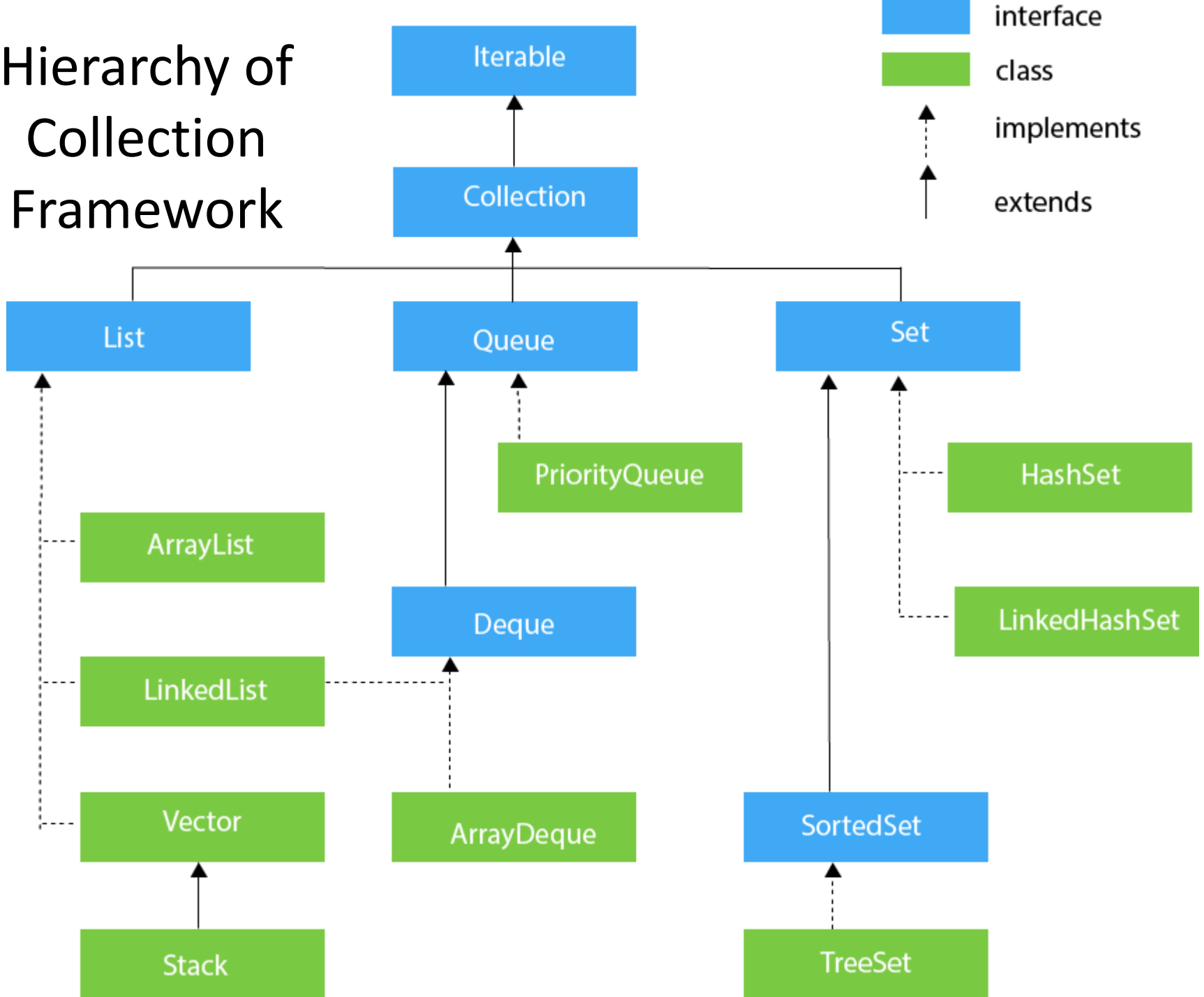
Collection Framework



Collection

- The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as [searching, sorting, insertion, manipulation, and deletion](#).
- Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Hierarchy of Collection Framework



Collection Interface - Methods

| Sr. | Method & Description |
|-----|--|
| 1 | <code>boolean add(E e)</code> It is used to insert an element in this collection. |
| 2 | <code>boolean addAll(Collection<? extends E> c)</code> It is used to insert the specified collection elements in the invoking collection. |
| 3 | <code>void clear()</code> It removes the total number of elements from the collection. |
| 4 | <code>boolean contains(Object element)</code> It is used to search an element. |
| 5 | <code>boolean containsAll(Collection<?> c)</code> It is used to search the specified collection in the collection. |
| 6 | <code>boolean equals(Object obj)</code> Returns true if invoking collection and obj are equal. Otherwise returns false. |
| 7 | <code>int hashCode()</code> Returns the hashcode for the invoking collection. |

Collection Interface - Methods

| Sr. | Method & Description |
|-----|---|
| 8 | <code>boolean isEmpty()</code> Returns true if the invoking collection is empty. Otherwise returns false. |
| 9 | <code>Iterator iterator()</code> It returns an iterator. |
| 10 | <code>boolean remove(Object obj)</code> Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false. |
| 11 | <code>boolean removeAll(Collection<?> c)</code> It is used to delete all the elements of the specified collection from the invoking collection. |
| 12 | <code>boolean retainAll(Collection<?> c)</code> It is used to delete all the elements of invoking collection except the specified collection. |
| 13 | <code>int size()</code> It returns the total number of elements in the collection. |

Collection Interface - Methods

| Sr. | Method & Description |
|-----|--|
| 14 | <p><code>Object[] toArray()</code></p> <p>Returns an array that contains all the elements stored in the invoking collection. The array elements are copies of the collection elements.</p> |

List Interface

- The **List** interface extends **Collection** and declares the behavior of a collection that stores a sequence of elements.
- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- List is a generic interface with following declaration

```
interface List<E>
```

where E specifies the type of object.

List Interface (example)

```
import java.util.*;
public class CollectionsDemo {
    public static void main(String[] args) {
        List a1 = new ArrayList();
        a1.add("Sachin");
        a1.add("Sourav");
        a1.add("Shami");
        System.out.println("ArrayList Elements");
        System.out.print("\t" + a1);

        List l1 = new LinkedList();
        l1.add("Mumbai");
        l1.add("Kolkata");
        l1.add("Vadodara");
        System.out.println();
        System.out.println("LinkedList Elements");
        System.out.print("\t" + l1);
    }
}
```

Here **ArrayList**
& **LinkedList**
implements **List**
Interface

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>java CollectionsDemo
ArrayList Elements
    [Sachin, Sourav, Shami]
LinkedList Elements
    [Mumbai, Kolkata, Vadodara]
G:\Darshan\Java 2019\PPTs\HAD\Programs>
```


List Interface - Methods

| Sr. | Method & Description |
|-----|---|
| 1 | <code>void add(int index, Object obj)</code> Inserts <code>obj</code> into the invoking list at the index passed in <code>index</code> . Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. |
| 2 | <code>boolean addAll(int index, Collection c)</code> Inserts all elements of <code>c</code> into the invoking list at the index passed in <code>index</code> . Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise. |
| 3 | <code>Object get(int index)</code> Returns the object stored at the specified index within the invoking collection. |
| 4 | <code>int indexOf(Object obj)</code> Returns the index of the first instance of <code>obj</code> in the invoking list. If <code>obj</code> is not an element of the list, <code>-1</code> is returned. |
| 5 | <code>int lastIndexOf(Object obj)</code> Returns the index of the last instance of <code>obj</code> in the invoking list. If <code>obj</code> is not an element of the list, <code>-1</code> is returned. |

List Interface (methods) (cont.)

| Sr. | Method & Description |
|-----|---|
| 6 | <code>ListIterator listIterator()</code> Returns an iterator to the start of the invoking list. |
| 7 | <code>ListIterator listIterator(int index)</code> Returns an iterator to the invoking list that begins at the specified index. |
| 8 | <code>Object remove(int index)</code> Removes the element at position <code>index</code> from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one |
| 9 | <code>Object set(int index, Object obj)</code> Assigns <code>obj</code> to the location specified by <code>index</code> within the invoking list. |
| 10 | <code>List subList(int start, int end)</code> Returns a list that includes elements from <code>start</code> to <code>end-1</code> in the invoking list. Elements in the returned list are also referenced by the invoking object. |

Iterator

- **Iterator** interface is used to cycle through elements in a collection, eg. displaying elements.
- **ListIterator** extends **Iterator** to allow bidirectional traversal of a list, and the modification of elements.
- Each of the collection classes provides an **iterator()** method that returns an iterator to the start of the collection. By using this iterator object, you can access each element in the collection, one element at a time.
- To use an iterator to cycle through the contents of a collection, follow these steps:
 1. Obtain an iterator to the start of the collection by calling the collection's **iterator()** method.
 2. Set up a loop that makes a call to **hasNext()**. Have the loop iterate as long as **hasNext()** returns true.
 3. Within the loop, obtain each element by calling **next()**.

Iterator - Example

```
import java.util.*;
public class IteratorDemo {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<String>();
        al.add("C");
        al.add("A");
        al.add("E");
        al.add("B");
        al.add("D");
        al.add("F");
        System.out.print("Contents of List: ");
        Iterator<String> itr = al.iterator();
        while(itr.hasNext()) {
            Object element = itr.next();
            System.out.print(element + " ");
        }
    }
}
```

```
G:\Darshan\Java 2019\PPTs\HAD\Pr
Contents of list: C A E B D F
```

Iterator - Methods

| Sr. | Method & Description |
|-----|---|
| 1 | boolean hasNext() Returns true if there are more elements. Otherwise, returns false. |
| 2 | E next() Returns the next element. Throws NoSuchElementException if there is not a next element. |
| 3 | void remove() Removes the current element. Throws IllegalStateException if an attempt is made to call remove() that is not preceded by a call to next() |

Comparator

- Comparator interface is used to set the sort order of the object to store in the sets and lists.
- The Comparator interface defines two methods: `compare()` and `equals()`.
- `int compare(Object obj1, Object obj2)`
obj1 and obj2 are the objects to be compared. This method returns zero if the objects are equal. It returns a positive value if obj1 is greater than obj2. Otherwise, a negative value is returned.
- `boolean equals(Object obj)`
obj is the object to be tested for equality. The method returns true if obj and the invoking object are both Comparator objects and use the same ordering. Otherwise, it returns false.

| | |
|--|---|
| <pre> import java.util.*; class Student { String name; int age; Student(String name, int age){ this.name = name; this.age = age; } } </pre> | <pre> class AgeComparator implements Comparator<Object>{ public int compare(Object o1,Object o2){ Student s1=(Student)o1; Student s2=(Student)o2; if(s1.age==s2.age) return 0; else if(s1.age>s2.age) return 1; else return -1; } } </pre> |
| <pre> public class ComparatorDemo { public static void main(String args[]){ ArrayList<Student> al=new ArrayList<Student>(); al.add(new Student("Vijay",23)); al.add(new Student("Ajay",27)); al.add(new Student("Jai",21)); System.out.println("Sorting by age"); Collections.sort(al,new AgeComparator()); Iterator<Student> itr2=al.iterator(); while(itr2.hasNext()){ Student st=(Student)itr2.next(); System.out.println(st.name+" "+st.age); } } } </pre> <div data-bbox="1207 721 1912 1092"> <p>G:\Darshan\Java 2019\</p> <p>Sorting by age</p> <p>Jai 21</p> <p>Vijay 23</p> <p>Ajay 27</p> </div> | |

Vector Class

- **Vector** implements a dynamic array.
- It is similar to **ArrayList**, but with two differences:
 - Vector is synchronized.
 - Vector contains many legacy methods that are not part of the collection framework
- Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.
- Vector is declared as follows:

```
Vector<E> = new Vector<E>;
```



```

import java.util.*;
public class VectorDemo {
    public static void main(String args[]) {
        //Create an empty vector with initial capacity 4
        Vector<String> vec = new Vector<String>(4);
        //Adding elements to a vector
        vec.add("Tiger");
        vec.add("Lion");
        vec.add("Dog");
        vec.add("Elephant");
        //Check size and capacity
        System.out.println("Size is: "+vec.size());
        System.out.println("Default capacity is: "+vec.capacity());
        //Display Vector elements
        System.out.println("Vector element is: "+vec);
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");
        //Again check size and capacity after two insertions
        System.out.println("Size after addition: "+vec.size());
        System.out.println("Capacity after addition is: "+vec.capacity());
        //Display Vector elements again
        System.out.println("Elements are: "+vec);
        //Checking if Tiger is present or not in this vector
        if(vec.contains("Tiger"))
        {
            System.out.println("Tiger is present at the index 0");
        }
        else
        {
            System.out.println("Tiger is not present in this vector");
        }
        //Get the first element
        System.out.println("The first animal of the vector is = " + vec.elementAt(0));
        //Get the last element
        System.out.println("The last animal of the vector is = " + vec.elementAt(vec.size()-1));
    }
}

```

```

G:\Darshan\Java 2019\PPTs\HAD\Programs>java VectorDemo
Size is: 4
Default capacity is: 4
Vector element is: [Tiger, Lion, Dog, Elephant]
Size after addition: 7
Capacity after addition is: 8
Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
Tiger is present at the index 0
The first animal of the vector is = Tiger
The last animal of the vector is = Deer

```

Vector - Constructors

| Sr. | Constructor & Description |
|-----|---|
| 1 | Vector() This constructor creates a default vector, which has an initial size of 10 |
| 2 | Vector(int size) This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size: |
| 3 | Vector(int size, int incr) This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward |
| 4 | Vector(Collection c) creates a vector that contains the elements of collection c |

Vector - Methods

| Sr. | Method & Description |
|-----|--|
| 7 | boolean containsAll (Collection c) Returns true if this Vector contains all of the elements in the specified Collection. |
| 8 | Enumeration elements () Returns an enumeration of the components of this vector. |
| 9 | Object firstElement () Returns the first component (the item at index 0) of this vector. |
| 10 | Object get (int index) Returns the element at the specified position in this Vector. |
| 11 | int indexOf (Object elem) Searches for the first occurrence of the given argument, testing for equality using the equals method. |
| 12 | boolean isEmpty () Tests if this vector has no components. |

Vector - Methods (cont.)

| Sr. | Method & Description |
|-----|---|
| 13 | Object lastElement() Returns the last component of the vector. |
| 14 | int lastIndexOf(Object elem) Returns the index of the last occurrence of the specified object in this vector. |
| 15 | Object remove(int index) Removes the element at the specified position in this Vector. |
| 16 | boolean removeAll(Collection c) Removes from this Vector all of its elements that are contained in the specified Collection. |
| 17 | Object set(int index, Object element) Replaces the element at the specified position in this Vector with the specified element. |
| 18 | int size() Returns the number of components in this vector. |

Stack

- **Stack** is a subclass of **Vector** that implements a standard last-in, first-out stack.
- **Stack** only defines the default constructor, which creates an empty stack.
- **Stack** includes all the methods defined by **Vector** and adds several of its own.
- **Stack** is declared as follows:

```
Stack<E> st = new Stack<E>();
```

where E specifies the type of object.

```

import java.util.*;
public class StackDemo {
    static void showpush(Stack<Integer> st, int a) {
        st.push(new Integer(a));
        System.out.println("push(" + a + ")");
        System.out.println("stack: " + st);
    }
    static void showpop(Stack<Integer> st) {
        System.out.print("pop -> ");
        Integer a = (Integer) st.pop();
        System.out.println(a);
        System.out.println("stack: " + st);
    }
    public static void main(String args[]) {
        Stack<Integer> st = new Stack<Integer>();
        System.out.println("stack: " + st);
        showpush(st, 42);
        showpush(st, 66);
        showpush(st, 99);
        showpop(st);
        showpop(st);
        showpop(st);
        try {
            showpop(st);
        } catch (EmptyStackException e) {
            System.out.println("empty stack");
        }
    }
}

```

```

G:\Darshan\Java 2019\PP1
stack: []
push(42)
stack: [42]
push(66)
stack: [42, 66]
push(99)
stack: [42, 66, 99]
pop -> 99
stack: [42, 66]
pop -> 66
stack: [42]
pop -> 42
stack: []
pop -> empty stack

```

Stack - Methods

- Stack includes all the methods defined by Vector and adds several methods of its own.

| Sr. | Method & Description |
|-----|--|
| 1 | <code>boolean empty()</code> Returns true if the stack is empty, and returns false if the stack contains elements. |
| 2 | <code>E peek()</code> Returns the element on the top of the stack, but does not remove it. |
| 3 | <code>E pop()</code> Returns the element on the top of the stack, removing it in the process. |
| 4 | <code>E push(E element)</code> Pushes element onto the stack. Element is also returned. |
| 5 | <code>int search(Object element)</code> Searches for element in the stack. If found, its offset from the top of the stack is returned. Otherwise, -1 is returned. |

Queue

- **Queue** interface extends **Collection** and declares the behaviour of a queue, which is often a first-in, first-out list.
- **LinkedList** and **PriorityQueue** are the two classes which implements Queue interface
- **Queue** is declared as follows:

```
Queue<E> q = new LinkedList<E>();  
Queue<E> q = new PriorityQueue<E>();
```

where E specifies the type of object.

Queue Example

```
G:\Darshan\Java 2019\PPTs\HAD\Programs>java QueueDemo
Elements in Queue:[Tom, Jerry, Mike, Steve, Harry]
Removed element: Tom
Head: Jerry
poll(): Jerry
peek(): Mike
Elements in Queue:[Mike, Steve, Harry]
```

```
q.add("Harry");
System.out.println("Elements in Queue:"+q);
System.out.println("Removed element: "+q.remove());
System.out.println("Head: "+q.element());
System.out.println("poll(): "+q.poll());
System.out.println("peek(): "+q.peek());
System.out.println("Elements in Queue:"+q);
```

```
}
```

```
}
```

Queue - Methods

| Sr. | Method & Description |
|-----|--|
| 1 | E <code>element()</code> Returns the element at the head of the queue. The element is not removed. It throws <code>NoSuchElementException</code> if the queue is empty. |
| 2 | boolean <code>offer(E obj)</code> Attempts to add <code>obj</code> to the queue. Returns <code>true</code> if <code>obj</code> was added and <code>false</code> otherwise. |
| 3 | E <code>peek()</code> Returns the element at the head of the queue. It returns <code>null</code> if the queue is empty. The element is not removed. |
| 4 | E <code>poll()</code> Returns the element at the head of the queue, removing the element in the process. It returns <code>null</code> if the queue is empty. |
| 5 | E <code>remove()</code> Returns the element at the head of the queue, returning the element in the process. It throws <code>NoSuchElementException</code> if the queue is empty. |

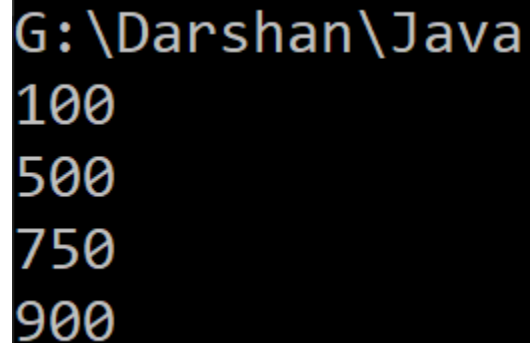
PriorityQueue

- `PriorityQueue` extends `AbstractQueue` and implements the `Queue` interface.
- It creates a queue that is prioritized based on the queue's comparator.
- `PriorityQueue` is declared as follows:

```
PriorityQueue<E> = new PriorityQueue<E>;
```
- It builds an empty queue with starting capacity as 11.

PriorityQueue - Example

```
import java.util.*;
public class PriorityQueueExample {
    public static void main(String[] args) {
        PriorityQueue<Integer> numbers = new PriorityQueue<>();
        numbers.add(750);
        numbers.add(500);
        numbers.add(900);
        numbers.add(100);
        while (!numbers.isEmpty()) {
            System.out.println(numbers.remove());
        }
    }
}
```

A terminal window showing the output of the Java program. The path 'G:\Darshan\Java' is at the top, followed by the numbers 100, 500, 750, and 900, which are the elements removed from the priority queue in ascending order.

```
G:\Darshan\Java
100
500
750
900
```

PriorityQueue - Constructors

| Sr. | Constructor & Description |
|-----|--|
| 1 | PriorityQueue() Creates a PriorityQueue with the default initial capacity (11) that orders its elements according to their natural ordering. |
| 2 | PriorityQueue(Collection<? extends E> c) Creates a PriorityQueue containing the elements in the specified collection. |
| 3 | PriorityQueue(int initialCapacity) Creates a PriorityQueue with the specified initial capacity that orders its elements according to their natural ordering. |
| 4 | PriorityQueue(int initialCapacity, Comparator<? super E> comparator) Creates a PriorityQueue with the specified initial capacity that orders its elements according to the specified comparator. |
| 5 | PriorityQueue(PriorityQueue<? extends E> c) Creates a PriorityQueue containing the elements in the specified priority queue. |
| 6 | PriorityQueue(SortedSet<? extends E> c) Creates a PriorityQueue containing the elements in the specified sorted set. |

PriorityQueue - Methods

| Sr. | Method & Description |
|-----|--|
| 1 | <code>boolean add(E e)</code> Inserts the specified element into this priority queue. |
| 2 | <code>void clear()</code> Removes all of the elements from this priority queue. |
| 3 | <code>Comparator<E> comparator()</code> Returns the comparator used to order the elements in this queue, or null if this queue is sorted according to the natural ordering of its elements. |
| 4 | <code>boolean contains(Object o)</code> Returns true if this queue contains the specified element. |
| 5 | <code>Iterator<E> iterator()</code> Returns an iterator over the elements in this queue. |
| 6 | <code>boolean offer(E e)</code> Inserts the specified element into this priority queue. |

PriorityQueue - Methods

| Sr. | Method & Description |
|-----|--|
| 7 | E peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty. |
| 8 | E poll() Retrieves and removes the head of this queue, or returns null if this queue is empty. |
| 9 | boolean remove(Object o) Removes a single instance of the specified element from this queue, if it is present. |
| 10 | int size() Returns the number of elements in this collection. |
| 11 | Object[] toArray() Returns an array containing all of the elements in this queue. |