Methods of PriorityQueue :
----
public boolean offer(E e) /public boolean add(E e) :- Used to add an element in the Queue.

public E poll() :- It is used to fetch the elements from head of the queue, after fetching it will delete the element.

public E peek() :- It is also used to fetch the elements from head of the queue, Unlike poll it will only fetch but not delete the element.

public boolean remove(Object element) :- It is used to remove an element. The return type is boolean.

public boolean isEmpty() : Used to verify whether the queue is empty or not.

//Programs :

```
package com.ravi.priority_queue;

import java.util.PriorityQueue;

public class PriorityQueueDemo1
{
    public static void main(String[] argv)
    {
            PriorityQueue<Object> pq = new PriorityQueue<>();
            pq.add("Orange");
            pq.add("Apple");
            pq.add("Mango");
            pq.add("Guava");
            pq.add("Grapes");

            //pq.add(null); // Inavlid
            //pq.add(23);     //Invalid
            System.out.println(pq);
    }

}
```

```
package com.ravi.priority_queue;

import java.util.PriorityQueue;

public class PriorityQueueDemo2
{
     public static void main(String[] argv)
     {
            PriorityQueue<Integer> pq = new PriorityQueue<>();
            pq.add(11);
            pq.add(2);
            pq.add(4);
            pq.add(6);
           System.out.println(pq);
     }
}
```

```
package com.ravi.priority_queue;

import java.util.Collections;
import java.util.PriorityQueue;

public class PriorityQueueDemo3
{
      public static void main(String[] args)
        {
            PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

            maxHeap.add(15);
            maxHeap.add(5);
            maxHeap.add(25);

            while (!maxHeap.isEmpty())
            {
                System.out.println(maxHeap.poll());
            }
        }
}
```

```
package com.ravi.priority_queue;

import java.util.PriorityQueue;

record Task(String name, Integer priority)
{
}


public class PriorityQueueDemo4
{
    public static void main(String[] args)
    {
        PriorityQueue<Task> taskQueue = new PriorityQueue<>
        ((t1,t2)-> t1.priority().compareTo(t2.priority()));

        taskQueue.add(new Task("Submit report", 4));
        taskQueue.add(new Task("Find Bug", 2));
        taskQueue.add(new Task("Write Program", 1));
        taskQueue.add(new Task("Execute Program", 3));

        while (!taskQueue.isEmpty())
        {
            System.out.println("Executing: " + taskQueue.poll());
        }
    }
}
```

```
package com.ravi.priority_queue;

import java.util.PriorityQueue;
import java.util.Scanner;

public class PriorityQueueDemo5
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        PriorityQueue<Integer> pq = new PriorityQueue<>(); // Min-Heap by default
        int choice;

        do
        {
            System.out.println("\n=== PriorityQueue Menu ===");
            System.out.println("1. Insert element");
            System.out.println("2. Remove head element (poll)");
            System.out.println("3. View head element (peek)");
            System.out.println("4. Display PriorityQueue");
            System.out.println("5. Exit");
            System.out.print("Enter your choice: ");
            choice = sc.nextInt();

            switch (choice)
    {
                case 1:
                    System.out.print("Enter element to insert: ");
                    int val = sc.nextInt();
                    pq.add(val);
                    System.out.println(val + " inserted.");
                    break;

                case 2:
                    if (pq.isEmpty()) {
                        System.out.println("PriorityQueue is empty!");
                    } else {
                        System.out.println("Removed: " + pq.poll());
                    }
                    break;

                case 3:
                    if (pq.isEmpty()) {
                        System.out.println("PriorityQueue is empty!");
                    } else {
                        System.out.println("Head element: " + pq.peek());
                    }
                    break;

                case 4:
                    System.out.println("PriorityQueue: " + pq);
                    break;

                case 5:
                    System.out.println("Exiting...");
                    break;

                default:
                    System.out.println("Invalid choice!");
            }
        } while (choice != 5);

        sc.close();
    }
}
```

Optional<T> :
----
* It is a final and immutable class available from JDK 1.8V.available in java.util package

* It is used to avoid NullPointerException in the industry, From JDK 1.5V onwards we are using Wrapper classes and user-defined classes (Avoiding the primitive) so chances of getting NullPointerException is very high.

Example :

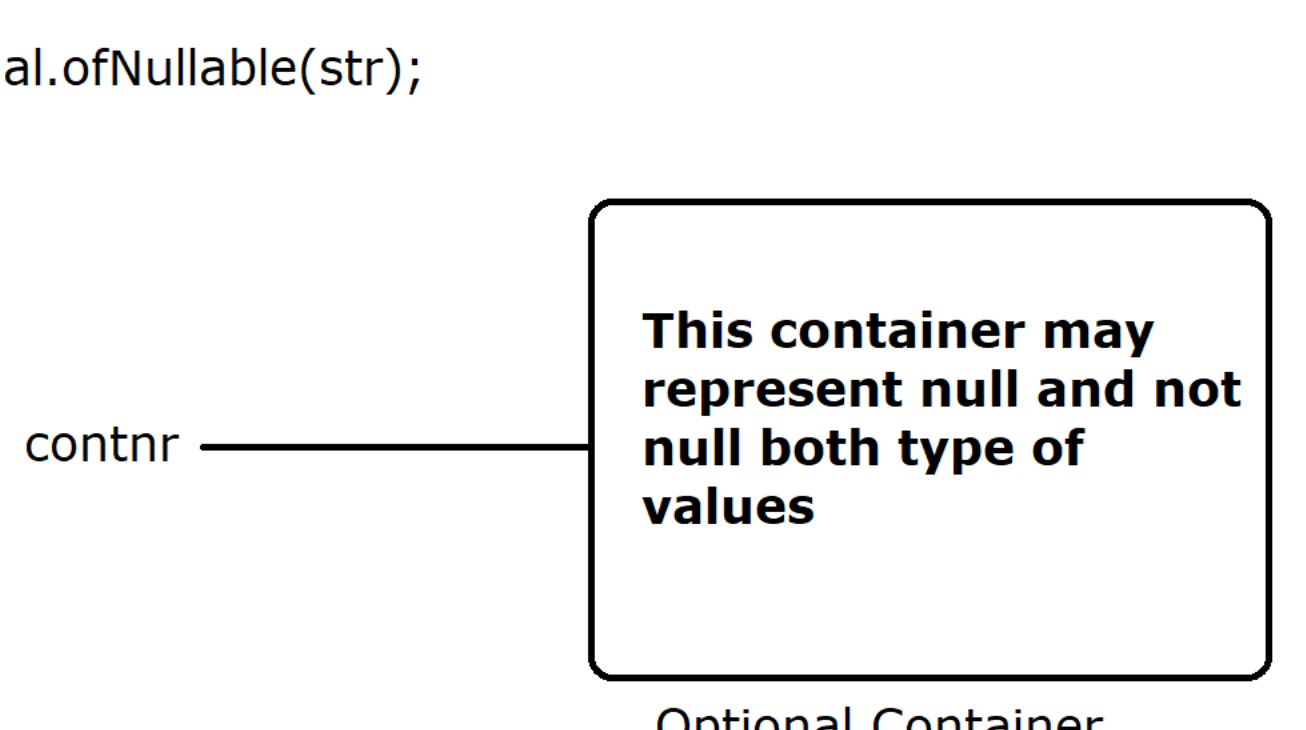
```
public class Student
{
   private Integer roll;

   public Integer getRoll()
   {
   }
}
```

```
Student s1 = new Student();
Integer roll =  s1.getRoll();

if(roll !=null)
{
   //perform operation
}
```

* Optional<String> contnr = Optional.ofNullable(str);



Optional Container

It is a container object which is used to represent an object (Optional object) that may or may not contain a non-null value.

If the value is available in the container, isPresent() method will return true and get() method will return the actual value.

Methods of Optional<T> class :
----
1) public static <T> Optional<T> ofNullable(T x) :

It will return the object of Optional class with specified value. If the specified value is null then this method will return an empty object of the optional class.

2) public boolean isPresent() :

It will return true, if the value is available in the container otherwise it will return false.

3) public T get() :

It will get/fetch the value from the container, if the value is not available then it will throw java.util.NoSuchElementException.

4) public T orElse(T defaultValue) :

It will return the value, if available otherwise it will return the specified default value.

5) public static Optional<T> of (T value) :

It will return the optional object with the specified value that is non- null value becuase it does not contain any container.

6) public static Optional<T> empty() :

It will return an empty Optional Object.

7) public java.util.stream.Stream stream()

It will Convert optional to Stream.

8) public void ifPresent(Consumer<T> cons) :

If a value is present, performs the given action with the value,
otherwise does nothing.

```
package com.ravi.optional;

import java.util.Optional;
import java.util.stream.Stream;

public class OptionalDemo1
{
    public static void main(String[] args)
    {
        String str = null;

        Optional<String> container = Optional.ofNullable(str);

        if(container.isPresent())
        {
            System.out.println("Value in the container :"+container.get());
        }
        else
        {
            System.err.println("No Value in the container");
        }

    }

}
```

```
package com.ravi.optional;

import java.util.Optional;

public class OptionalDemo2
{
    public static void main(String[] args)
    {
        Integer i = null;

        Optional<Integer> container = Optional.ofNullable(i);
         System.out.println("Value in the container :"+container.get());

    }

}
```

```
package com.ravi.optional;

import java.util.Optional;

public class OptionalDemo3
{
    public static void main(String[] args)
    {
        String str = null;

        Optional<String> container = Optional.ofNullable(str);

        String orElse = container.orElse("No Value in the container");

        System.out.println(orElse);

    }

}
```

```
package com.ravi.optional;

import java.util.Optional;

class Employee
{
    private Integer id;
    private String name;

    public Employee()
     {
        super();

    }

    public Employee(Integer id, String name)
    {
        super();
        this.id = id;
        this.name = name;
    }

     //New getter style from JDK 1.8V

    public Optional<Integer> getId()
    {
        return Optional.ofNullable(id);
    }

    public Optional<String> getName()
    {
        return Optional.ofNullable(name);
    }

    @Override
    public String toString()
    {
        return "Employee [id=" + id + ", name=" + name + "]";
    }

}


public class OptionalDemo4
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        e1 = new Employee(111, "Scott");

        Optional<String> name = e1.getName();
        System.out.println(name.orElse("Name is not available"));

         Optional<Integer> id = e1.getId();
          id.ifPresent(System.out::println);

    }

}
```

```
package com.ravi.optional;

import java.util.ArrayList;
import java.util.Optional;

public class OptionalDemo5
{
    public static void main(String[] args)
    {
        ArrayList<Optional<String>> listOfCity = new ArrayList<>();
         listOfCity.add(Optional.of("Hyd"));
         listOfCity.add(Optional.of("Pune"));
         listOfCity.add(Optional.of("Indore"));
         listOfCity.add(Optional.of("Chennai"));
         listOfCity.add(Optional.empty());

         for(Optional<String> optnl : listOfCity)
         {
                if(optnl.isPresent())
                {
                    System.out.println(optnl.get());
                }
                else
                {
                    System.err.println("Value is not available");
                }
         }

    }

}
```