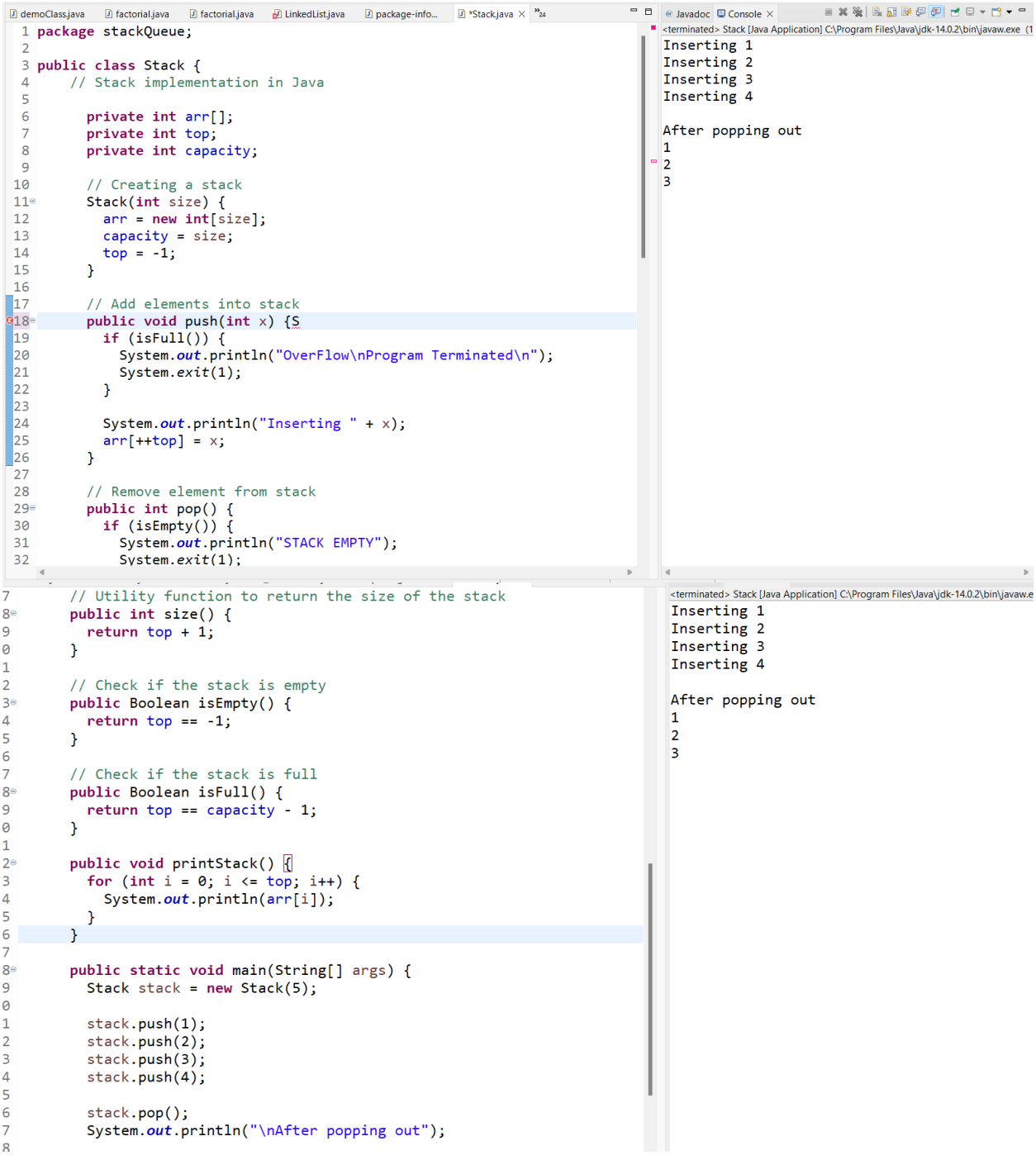
Stack :

```
package stackQueue;

public class Stack {
     // Stack implementation in Java

       private int arr[];
       private int top;
       private int capacity;

       // Creating a stack
       Stack(int size) {
         arr = new int[size];
         capacity = size;
         top = -1;
       }

       // Add elements into stack
       public void push(int x) {
         if (isFull()) {
           System.out.println("OverFlow\nProgram
Terminated\n");
           System.exit(1);
         }

         System.out.println("Inserting " + x);
         arr[++top] = x;
       }

       // Remove element from stack
       public int pop() {
         if (isEmpty()) {
           System.out.println("STACK EMPTY");
           System.exit(1);
         }
         return arr[top--];
       }
```

```java
  // Utility function to return the size of the stack
  public int size() {
    return top + 1;
  }

  // Check if the stack is empty
  public Boolean isEmpty() {
    return top == -1;
  }

  // Check if the stack is full
  public Boolean isFull() {
    return top == capacity - 1;
  }

  public void printStack() {
    for (int i = 0; i <= top; i++) {
      System.out.println(arr[i]);
    }
  }

  public static void main(String[] args) {
    Stack stack = new Stack(5);

    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);

    stack.pop();
    System.out.println("\nAfter popping out");

    stack.printStack();

  }
}
```



```
demoClass.java  factorial.java  factorial.java  LinkedList.java  package-info...  *Stack.java

1  package stackQueue;
2
3  public class Stack {
4      // Stack implementation in Java
5
6        private int arr[];
7        private int top;
8        private int capacity;
9
10       // Creating a stack
11       Stack(int size) {
12         arr = new int[size];
13         capacity = size;
14         top = -1;
15       }
16
17       // Add elements into stack
18       public void push(int x) {S
19         if (isFull()) {
20           System.out.println("OverFlow\nProgram Terminated\n");
21           System.exit(1);
22         }
23
24         System.out.println("Inserting " + x);
25         arr[++top] = x;
26       }
27
28       // Remove element from stack
29       public int pop() {
30         if (isEmpty()) {
31           System.out.println("STACK EMPTY");
32           System.exit(1);
```

```
Javadoc  Console
<terminated> Stack [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (1
Inserting 1
Inserting 2
Inserting 3
Inserting 4

After popping out
1
2
3
```

```
7        // Utility function to return the size of the stack
8        public int size() {
9          return top + 1;
0        }
1
2        // Check if the stack is empty
3        public Boolean isEmpty() {
4          return top == -1;
5        }
6
7        // Check if the stack is full
8        public Boolean isFull() {
9          return top == capacity - 1;
0        }
1
2        public void printStack() {
3          for (int i = 0; i <= top; i++) {
4            System.out.println(arr[i]);
5          }
6        }
7
8        public static void main(String[] args) {
9          Stack stack = new Stack(5);
0
1          stack.push(1);
2          stack.push(2);
3          stack.push(3);
4          stack.push(4);
5
6          stack.pop();
7          System.out.println("\nAfter popping out");
8
```

```
<terminated> Stack [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.e
Inserting 1
Inserting 2
Inserting 3
Inserting 4

After popping out
1
2
3
```

QUEUE:

```
package stackQueue;

public class Queue {
     // Queue implementation in Java


        int SIZE = 5;
        int items[] = new int[SIZE];
        int front, rear;

        Queue() {
          front = -1;
          rear = -1;
        }

        boolean isFull() {
          if (front == 0 && rear == SIZE - 1) {
            return true;
          }
          return false;
        }

        boolean isEmpty() {
          if (front == -1)
            return true;
          else
            return false;
        }

        void enQueue(int element) {
          if (isFull()) {
            System.out.println("Queue is full");
          } else {
            if (front == -1)
              front = 0;
            rear++;
            items[rear] = element;
```

```
      System.out.println("Inserted " + element);
    }
  }

  int deQueue() {
    int element;
    if (isEmpty()) {
      System.out.println("Queue is empty");
      return (-1);
    } else {
      element = items[front];
      if (front >= rear) {
        front = -1;
        rear = -1;
      } /* Q has only one element, so we reset the queue
after deleting it. */
      else {
        front++;
      }
      System.out.println("Deleted -> " + element);
      return (element);
    }
  }

  void display() {
    /* Function to display elements of Queue */
    int i;
    if (isEmpty()) {
      System.out.println("Empty Queue");
    } else {
      System.out.println("\nFront index-> " + front);
      System.out.println("Items -> ");
      for (i = front; i <= rear; i++)
        System.out.print(items[i] + "  ");

      System.out.println("\nRear index-> " + rear);
    }
  }
```

```
    public static void main(String[] args) {
      Queue q = new Queue();

      // deQueue is not possible on empty queue
      q.deQueue();

      // enQueue 5 elements
      q.enQueue(1);
      q.enQueue(2);
      q.enQueue(3);
      q.enQueue(4);
      q.enQueue(5);

      // 6th element can't be added to because the queue
is full
      q.enQueue(6);

      q.display();

      // deQueue removes element entered first i.e. 1
      q.deQueue();

      // Now we have just 4 elements
      q.display();

    }
  }
```

```java
package stackQueue;

public class Queue {
    // Queue implementation in Java


      int SIZE = 5;
      int items[] = new int[SIZE];
      int front, rear;

      Queue() {
        front = -1;
        rear = -1;
      }

      boolean isFull() {
        if (front == 0 && rear == SIZE - 1) {
          return true;
        }
        return false;
      }

      boolean isEmpty() {
        if (front == -1)
          return true;
        else
          return false;
      }


    void enQueue(int element) {
      if (isFull()) {
        System.out.println("Queue is full");
      } else {
        if (front == -1)
          front = 0;
        rear++;
        items[rear] = element;
        System.out.println("Inserted " + element);
      }
    }

    int deQueue() {
      int element;
      if (isEmpty()) {
        System.out.println("Queue is empty");
        return (-1);
      } else {
        element = items[front];
        if (front >= rear) {
          front = -1;
          rear = -1;
        } /* Q has only one element, so we reset the queue after delet
        else {
          front++;
        }
        System.out.println("Deleted -> " + element);
        return (element);
      }
    }
```

```
<terminated> Queue [Java Application] C:\Program Files\Java\j
Queue is empty
Inserted 1
Inserted 2
Inserted 3
Inserted 4
Inserted 5
Queue is full

Front index-> 0
Items ->
1  2  3  4  5
Rear index-> 4
Deleted -> 1

Front index-> 1
Items ->
2  3  4  5
Rear index-> 4
```


```
Queue is empty
Inserted 1
Inserted 2
Inserted 3
Inserted 4
Inserted 5
Queue is full

Front index-> 0
Items ->
1  2  3  4  5
Rear index-> 4
Deleted -> 1

Front index-> 1
Items ->
2  3  4  5
Rear index-> 4
```