```java
//@author maximilian raspe
import java.util.Arrays;

public class Permutation {

   public static void main(String[] args) {
      int n = 4;
      int[] array = new int[n];
      arrayFuellen(array);
      permutationAusgeben(array, n);
   }
   //füllt den array mit zahlen
   private static void arrayFuellen(int[] array) {
      for(int i = 0; i < array.length; i++) {
         array[i] = i + 1;
      }
   }
   private static void permutationAusgeben(int[] array, int n) {
      //sobald n= 1 erreicht wurde wird der array ausgegeben
      if(n == 1) {
         System.out.println(Arrays.toString(array));
      }
      //array wird getauscht und ausgegeben
      for(int i = 0; i < n; i++) {
         tauschen(array, i, n - 1);
         permutationAusgeben(array, n - 1);
         tauschen(array, i, n - 1);
      }

   }
   //tauscht ints an index i und j
   private static void tauschen(int[] array, int i, int j) {
      int k = array[i];
      array[i] = array[j];
      array[j] = k;
   }
}

//@author maximilian raspe
public class Binomial {

   public static void main(String[] args) {
      binomialFakultat(8, 4);
      binomialProduktQuotient(8, 4);
      binomialAlternierend(8,4);
      binomialrekursiv(8, 4);

   }
   private static long binomialFakultat (int n, int k) {
      long ergebnis;
      ergebnis = fak(n) / (fak(k) * fak(n - k));
      return ergebnis;
```

```
}
private static long fak(int f) {
   long ergebnis = 0;
   if(f == 1) {
      return 1;
   }
   if(f == 2) {
      return 2;
   }
   if(f > 2) {
      ergebnis = f * fak(f - 1);
      return ergebnis;
   }
   return ergebnis;
}
private static long binomialProduktQuotient(int n, int k) {

   int zaehler = 1;
   long nenner;

   for(int i = n; i > n - k + 1; i--) {
      zaehler *= i;
   }
   zaehler = zaehler * (n - k + 1);
   nenner = fak(k);
   long ergebnis =  zaehler / nenner;

   return ergebnis;
}
private static long binomialAlternierend(int n, int k) {
   double zaehler = 1;
   double nenner = 1;
   long ergebnis = 1;
   int hilfsvariable = 1;

   for(int i = n; i >= n - k + 1; i--) {
      zaehler = i;
      nenner = hilfsvariable;
      ergebnis *= zaehler / nenner;
      hilfsvariable++;
   }
   return ergebnis;
}
private static long binomialrekursiv(int n, int k) {
   long ergebnis;
   if(k == 0) {
      return 1;
   }
   else if(n == 0) {
      return 0;
   }
   else {
```

```
            return binomialrekursiv(n - 1, k - 1) + binomialrekursiv(n - 1, k);
        }
    }
}

//@author maximilian raspe
public class Main {
    public static void main(String[] args) {
        DoubleLinkedList list = new DoubleLinkedList();
        list.add(5);
        list.add(7);
        list.add(3);
        list.add(2);
        list.add(9);
        list.insert(5, 3);
        list.delete(3);
        list.get(2);
        list.length();
        System.out.println(list.toString());
    }
}

//@author maximilian raspe
public class DoubleLinkedList<T> implements List<T> {

    private static class Node<T> {
        private T data;
        private Node<T> next;
        private Node<T> previous;
        //konstruktor für neuen knoten
        public Node(T data, Node previous, Node next) {
            this.data = data;
            this.previous = previous;
            this.next = next;
        }
    }

    private Node <T> current;
    private Node<T> tail;
    private Node<T> head;
    private int counter;

    //fügt das element am ende der liste an
    @Override
    public void add(T t)  {
        try {
            Node node = new Node(t, null, null);
            if(head == null) {
                head = node;
                head.previous = null;
                head.next = null;
            }
```

```
            else {
                current = head;

                while(current.next != null) {

                }
                current.next = node;
                node.previous = current;
                node.next = null;
            }
        } catch(Exception e) {
            System.out.println("Fehler" + e.getMessage());
        }

    }
    //fügt das gewünschte objekt an i-ter stelle ein. die schleife wird solange durchlaufen bis man am gewünschten index angekommen ist und dort wird das element eingefügt
    @Override
    public void insert(T obj, int index){
        try {
            Node node = new Node(obj, null, null);
            Node<T> temp;
            int iteration = 1;

            head = node;
            while (iteration != index) {

                iteration++;

            }
            temp = current.next;
            node.next = temp;
            temp.previous = node;
            node.previous = current;
        } catch(Exception e) {
            System.out.println("Fehler" + e.getMessage());
        }
    }
    //durchläuft die liste solange bis man am gewünschten index angekommen ist und gibt dann das element der liste zurück
    @Override
    public T get(int index){
        try {
            Node current = head;
            counter = 0;

            while (current != null) {
                if (counter == index) {
                    return (T) current.data;
                }
                counter++;
                current = current.next;
```

```
            }
        }
        catch(Exception e) {
            System.out.println("Fehler : " + e.getMessage());
        }
        return null;
    }
    //löscht das element an stelle vom index.
    @Override
    public void delete(int index) {
        try {
            int iterator = 1;

            current = head;
            if (index == 1) {
                head = current.next;
                current.next = null;
                current.previous = null;
            } else {
                while (iterator != index) {
                    current = current.next;
                    iterator++;
                }
                if (current.next == null) {
                    current.previous.next = null;
                    current.previous = null;
                } else {
                    current.previous.next = current.next;
                    current.next.previous = current.previous;
                }
            }
        } catch(Exception e) {
            System.out.println("Fehler" + e.getMessage());
        }
    }
    //itieriert solange durch die liste bis ein objekt gefunden wurde, wenn nicht wird nichts
zurückgegeben
    @Override
    public int indexOf(T obj) {
        try {
        for(int i = 1; i < counter; i++) {
            if (obj.equals(current.data)) {
                return i;
            }
            current = current.next;
        }

        } catch (Exception e) {
            System.out.println("Fehler : " + e.getMessage());
        }
        return -1;
    }
```

```
    //gibt die länge der liste an, wird mit counter gezählt
    @Override
    public int length() {
        return counter;
    }

    //gibt den inhalt des "knotens" als string zurück
    @Override
    public String toString() {
        Node node = head;
        while(node != null) {
            System.out.println(" " + node.data);
            node = node.next;
        }
        return "";
    }

    @Override
    public boolean equals(Object obj) {
        if(!(obj instanceof Node)) {
            return false;
        }
        Node<T> element = (Node<T>) obj;
        if(current.data == null) {
            return element.data == null;
        }
        return element.equals(element.data);
    }
}

//@author maximilian raspe
public interface List<T> {

    public void add(T obj);
    public void insert(T obj , int index);
    public T get(int index);
    public void delete(int index);
    public int indexOf(T obj);
    public int length ();
}
```