工程框架：采用工厂模式，各种全排列生成算法均实现了接口PermutationGeneration并且继承了Permutation基类。

其中接口PermutationGeneration里面有三类方法，分别为void genPermutation()，用于根据给定的初始序列按某种顺序生成之后的所有排列；void genPermutation(int pos)则是用于根据给定的初始序列按某种顺序生成之后的第pos个排列；void genAllPermutation(int per\_size)用于按照某种顺序生成per\_size大小的所有排列。Permutation基类包含了排列列表和排列长度属性，以及相关的初始化和输出操作。工厂类为PermutationFactory用于根据用户的选择生成不同的方法实例。Util是工具类，里面静态实现了计算十进制数的阶乘，以及转化为递增进位制数和递减进位制数等操作。PermutationTest是测试类，用户首先输入选择的全排列生成算法的序号，接着输入生成排列的长度，控制台即会一次输出所有的全排列。各全排列算法思路如下：

1. 字典序法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递增进位制表示的中介数(记录当前数字右边比当前数字小的数字个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从左往右依次填入，注意左边已经出现的比前数字小的数字)。

1. 递增进位制数法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递增进位制表示的中介数(anan-1…a2)↑, *ai*：*i*的右边比*i*小的数字的个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置,依次填入)。

1. 递减进位制数法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递减进位制表示的中介数((a2a3…an-1an)↓，*ai*：*i*的右边比*i*小的数字的个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置,依次填入)。

1. 邻位对换法法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递减进位制表示的中介数以及各位数的方向(2的方向一定是向左, 对于每一个大于2的数字*i*，如果i为奇数，其方向性决定于bi-1的奇偶性，奇向右、偶向左。如果i为偶数，其方向性决定于bi-1+ bi-2的奇偶性，同样是奇向右、偶向左。当得到方向性后，bi的值就是背向i的方向直到排列边界这个区间里比i小的数字的个数), 之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置以及方向,依次填入)。

公共函数伪代码：

//根据传入的排列长度初始化为升序排列

InitPermutationFromSize(size)

for i = 1 to size

permutation[i] = i

return permutation

字典序法伪代码：

LexOrder-Permutation(size)

permutation = InitPermutationFromSize(size)

mediaNumber = GetLexMediaNumber(permutation)

while(LexEnd(permutation) == false)

AddIncMediaNumber(mediaNumber, 1)

ConvertToLexPermutation(mediaNumber, permutation)

//获取字典序的递增进位制中介数

GetLexMediaNumber(permutation)

for i = 1 to permutation.size - 1

mediaNumber[i] = permutation中permuation[i]的右边比他小的数字的个数

return mediaNumber

//判断是否是字典序的最后一个排列

LexEnd(permutation)

for i = 1 to permutation.size-1

if permutation[i] < permutation[i+1]

return false

return true

//中介数转化为字典序排列

ConvertToLexPermutation(mediaNumber, permutation)

for i = 1 to permutation.size-1

permutation[i] = mediaNumber[i]+1

while未填入的数字中比permutation[i]小的数字个数小于mediaNumber[i]

permutation[i] = permutation[i] + 1

递增进位制数法伪代码：

IncOrder-Permutation(size)

permutation = InitPermutationFromSize(size)

mediaNumber = GetIncMediaNumber(permutation)

while(IncEnd(permutation) == false)

//返回中介数加1之后的递增进位制数表示

meidiaNumber = AddIncMediaNumber(mediaNumber, 1)

ConvertToIncPermutation(mediaNumber, permutation)

//获取递增进位制数法的中介数

GetIncMediaNumber(permutation)

for i = permutation.size to 2

mediaNumber[i] = 在permutation中i 的右边比它小的数字的个数

return mediaNumber

//判断是否为最后一个中介数

IncEnd(medianNumber)

for i = 2 to permutation.size

if mediaNumber[i] != (i-1)

return false

return true

//中介数转化为递增进位制数法排列

ConvertToIncPermutation(mediaNumber, permutation)

for i = permutation.size to 2

将i 放到permutation从右至左第mediaNumber[i]+1个空位上

将 数字1放在permutation剩下的空位上

递减进位制数法伪代码：

DecOrder-Permutation(size)

permutation = InitPermutationFromSize(size)

mediaNumber = GetDecMediaNumber(permutation)

while(DecEnd(mediaNumber) == false)

//返回中介数加1之后的递减进位制数表示

mediaNumber = AddDecMediaNumber(mediaNumber, 1)

ConvertToDecPermutation(mediaNumber, permutation)

//获取递减进位制数的中介数

GetDecMediaNumber(permutation)

for i = 2 to permutation.size

mediaNumber[i] = 在permutation中i的右边比它小的数字的个数

return mediaNumber

//判断是否为最后一个中介数

DecEnd(mediaNumber)

for i = 2 to permutation.size

if mediaNumber[i] != (i-1)

return false

return true

//中介数转化为递减进位制数法排列

ConvertToDecPermutation(mediaNumber, permutation)

for i = permutation.size downto 2

将i放到permutation从右至左第mediaNumber[i]+1个空位上

将 数字1放在permutation剩下的空位上

邻位对换法伪代码：

SJTOrder-Permutation(size)

permutation = InitPermutationFromSize(size)

GetMediaNumberAndDirection(permutation, mediaNumber, direction)

while(SJTEnd(permutation, direction) == false)

AddSJTMediaNumber(mediaNumber, 1, direction)

ConvertToSJTPermutation(mediaNumber, direction, permutation)

//获取邻位对换法的递减进位制中介数以及方向

GetMediaNumberAndDirection(permutation, mediaNumber, direction)

direction[2] = left

mediaNumber[2] = permutation中背向2的方向，即2的右边比2小的数字个数

for i = 3 to permutation.size

if i%2 == 1

if mediaNumber[i-1]%2==1

direction[i] = right

else

direction[i] = left

else

if (mediaNumber[i-1]+mediaNumber[i-2])%2==1

direction[i]=right

else

direction[i]=left

mediNumber[i] = permutation中背向i的方向至排列边界比i小的数字的个数

//判断是否是最后一个SJT排列

SJTEnd(permutation, direction)

for i = 1 to permutation.size

if permutation[i] != 1

if direction[i]==left

if i-1 < 1 or permutation[i-1] < permutation[i]

return false

else

if i+1 > permutation.size or permutation[i+1] < permutation[i]

return false

return true

//计算加1之后的中介数以及方向

AddSJTMediaNumber(mediaNumber, 1, direction)

将mediaNumber按递减进位制法加1

direction[2] = left

mediaNumber[2] = permutation中背向2的方向，即2的右边比2小的数字个数

for i = 3 to permutation.size

if i%2 == 1

if mediaNumber[i-1]%2==1

direction[i] = right

else

direction[i] = left

else

if (mediaNumber[i-1]+mediaNumber[i-2])%2==1

direction[i]=right

else

direction[i]=left

//中介数转化为邻位对换法的排列

ConvertToSJTPermutation(mediaNumber, direction, permutation)

for i = permutation.size downto 2

if direction[i] == left

将i放到permutation从右至左的第mediaNumber[i]+1个空位上

else

将i放到permutation从左至右的第mediaNumber[i]+1个空位上

将数字1放到permutation剩下的空位上