Starting with this challenge, one of the most subtle changes is the following:

Previous challenge:
```
if (s.substring(startPos,(startPos+p.length())).indexOf(sb.toString().charAt(pos))!=-1)
```

This Challenge:
```
if (s.substring(startPos,(startPos+p.length())).indexOf(sb.toString().charAt(pos))!=0)
```

It has to be realised that in previous challenge we were not concerned if characters in String p appeared in any order in the block of String s.

In this new challenge, we require the match to be at index 0.
The reason is since the challenge required all permutations to be explored.
If we were to not make the above change, we would expect that ALL permutations to either appear in String s OR none of the permutations to appear in String s.
This is not what the challenge is trying to achieve!

```java
for (int i=startPos; i<s.length();i++)
{
    if(!sb.toString().isEmpty())
    {
        System.out.println("\nThis is STARTPOS: " + startPos);
        System.out.println("This is sb length: " + sb.length());
    }

    for (int pos=0; pos<=sb.length(); pos++)
    {
```

We also need to remove this since it also contradicts the permutation. Instead we need to increment the pos variable.
And break out the outer for loop when pos>=p.length()

```java
            pos++;

            if (pos==p.length())
            {
                break;
            }
        //}
    }
```

Also need to modify this section since we are no longer interested in substring of String s..

```java
//now the focus is on checking to see if the character of String p appears in that order in block size in String s
if (Character.toString(s.charAt(i)).indexOf(sb.toString().charAt(pos))==0)
```

Whilst I am continuing with this challenge, the first real issue occurred here.

```
194
195                 //for (int pos=0; pos<=sb.length(); pos++)
196                 //{
197                     if (hasCharFound)
198                     {
199                         pos=0;
200                     }
201                     if (!sb.toString().isEmpty())
202                     {
203                         System.out.println("value of pos: " + pos);
204                         System.out.println("value of sb: " + sb.toString());
205                         System.out.println("Checking character: " + sb.toString().charAt(pos) +
206                         "  against the main String index: " + i + "("+s.charAt(i)+")");
207
208                         System.out.println("SUBSTRING EXAMINED: " + s.substring(startPos,(startPos+p.length())));
209
210                         //now the focus is on checking to see if the character of String p appears in that order in block size in String s
211                         if (Character.toString(s.charAt(i)).indexOf(sb.toString().charAt(pos))==0)
212                         {
```

Ln: 208, Col: 118

Run | Share | Command Line Arguments

```
This is STARTPOS: 8
This is sb length: 3
value of pos: 0
value of sb: bcb
Checking character: b  against the main String index: 8(c)
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 8, end 11, length 10
   at java.base/java.lang.String.checkBoundsBeginEnd(String.java:3319)
   at java.base/java.lang.String.substring(String.java:1874)
   at Main.findAnagrams(Main.java:208)
   at Main.main(Main.java:52)
```

I am finding this error something I did not expect

I have not changed anything in my code to

```
*****RESTORING BACKUP OF STRINGBUILDER (String p): abc

This is STARTPOS: 7
This is sb length: 3
value of pos: 0
value of sb: abc
Checking character: a  against the main String index: 7(a) TO index: 9(d)
SUBSTRING EXAMINED: acd
char found: a    at index: 0
a has been removed from StringBuilder (String p)= abc
This is current StringBuilder (String p): bc
value of pos: 0
value of sb: bc
Checking character: b  against the main String index: 7(a) TO index: 9(d)
SUBSTRING EXAMINED: acd
NO MATCH FOUND
StringBuilder being emptied: bc
*****RESTORING BACKUP OF STRINGBUILDER (String p): abc
```

If we examine the previous challenge it had the intelligence to stop at 7 when processing the same test condition

```
15          //TEST CASE 1:
16          findAnagrams("cbaebabacd", "abc");
```

Strictly speaking, using the do while loop in my first code, it should have also failed to progress since I have not modified these...
But we know the structure is technically correct:

```
}while(p.length()+startPos<s.length());
```

We can see that this is post test evaluation

We can clearly see that

p.length()=3

startPos= 7

s.length= 10

```
NO MATCH FOUND
StringBuilder being emptied: ba
*****RESTORING BACKUP OF STRINGBUILDER (String p): bba

This is STARTPOS: 7
This is sb length: 3
This is s length: 10
value of pos: 0
value of sb: bba
Checking character: b  against the main String index: 7(a)
SUBSTRING EXAMINED: acd
NO MATCH FOUND
StringBuilder being emptied: bba
*****RESTORING BACKUP OF STRINGBUILDER (String p): bba
aaa    Subset: 2  at cycle number: 6
THIS IS SB: aaa

This is STARTPOS: 8
This is sb length: 3
This is s length: 10
value of pos: 0
```

There is absolutely no reason for the code to reach here...

I will keep this code in my repository.

Perhaps I will opt for a while loop instead

```
178
179                 while(p.length()+startPos<s.length())
180         {
181             do
182             {
183                 startPos=counter;
184
185                 for (int i=startPos; i<s.length();i++)
```

Ln: 174, Col: 69

Run | Share | Command Line Arguments

```
This is STARTPOS: 7
This is sb length: 3
This is s length: 10
value of pos: 0
value of sb: bba
Checking character: b  against the main String index: 7(a)
SUBSTRING EXAMINED: acd
NO MATCH FOUND
StringBuilder being emptied: bba
*****RESTORING BACKUP OF STRINGBUILDER (String p): bba
abb    Subset: 2  at cycle number: 7
THIS IS SB: abb
aba    Subset: 3  at cycle number: 7
THIS IS SB: aba
acb    Subset: 4  at cycle number: 7
THIS IS SB: acb
cba    Subset: 5  at cycle number: 7
THIS IS SB: cba
aca    Subset: 6  at cycle number: 7
THIS IS SB: aca

** Process exited - Return Code: 0 **
```

My next aim is to determine why it has not processed other scenarios in valuesSet of ALL permutations and just skipped them!

The program has terminated properly

In regards to investigating the next permutation, I would need to simply look at the area of code in question...
Logic suggests it has to be related to the following condition.

```
while(p.length()+startPos<s.length())
```

We know from the previous challenge, once it traversed through String s, there was no need to start the process again from the offset.
We know with this challenge, we still have outstanding permutations...
So logic suggests that once it exits out of the inner do while loop, we would need to set startPos back to 0 and also counter to 0

I have completed the following change:

```
    } //end of while loop  while(p.length()+startPos<s.length())

      //but we will find that counter is still one greater than startPos in which the while loop terminated
    // so effectively, startPos will once again become 8.
    //and for a main String s where the length is 10 characters, this will invalidate the situation and generate a
    //StringIndexOutOfBoundsException
    //effectively I have set the counter =0  once it exits the while loop.
    //we know at this point it is in the for loop where it is processing the permutations.
    //I believe all the conditions are now set for a fresh execution....

    counter=0;
    startPos=0;

```

And finally it runs through all my test executions...
Now clearly, it would be best that I compose my test case String with all the permutations...

However as I reached the end of the challenge, I had realised that I have made a fundamental mistake during the process of getting the permutations.
I had clearly forgotten to remove the selection.

Infact my selection was still taking place via random index from a String. So the only route forward was to configure a List with characters from String p.
And attempt selection.

```
*****************Contents of the backup set
*****************Contents of the valuesSet
Subset 1: abb
Subset 2: acb
Subset 3: cbb
Subset 4: bcc
Subset 5: bbb
Subset 6: bac
*************NEW VALUE CYCLES: 6
*************RUNNING TOTAL CYCLES: 6
***PROCESSING SET AT INDEX: 0
```

THIS IS EQUIVALENT TO PERMUTATION WITHOUT REPLACEMENT AND IT WAS NOT THE NATURE OF THE CHALLENGE SINCE ALL LETTERS HAD TO BE USED EXACTLY ONCE AND DISCARDED AGAIN FROM SELECTION

Instead of detailing walkthrough, I have annotated my code and quite efficiently applied necessary changes.
It once again caused confusion as to use Lists and how to perform random selection without replication of indexes. I eventually reached a solution.
And it can be seen that selection is now as expected...

```
******************Contents of the backup set
******************Contents of the valuesSet
Subset 1: bca
Subset 2: acb
Subset 3: abc
Subset 4: cba
Subset 5: cab
Subset 6: bac
*************NEW VALUE CYCLES: 16
```

It will be difficult to present test cases, however I will perform a full output of all the screen messages...