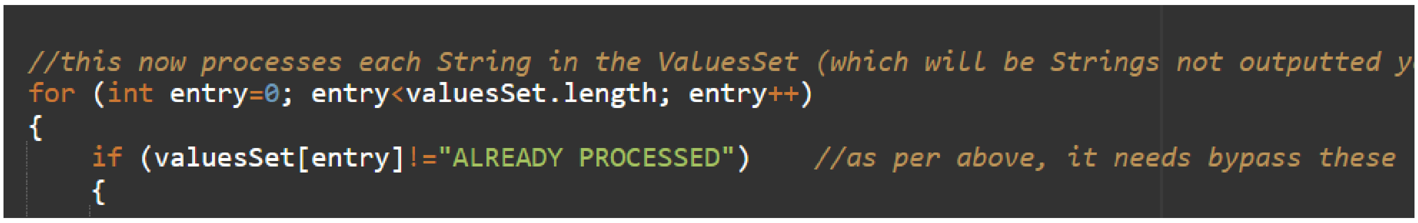
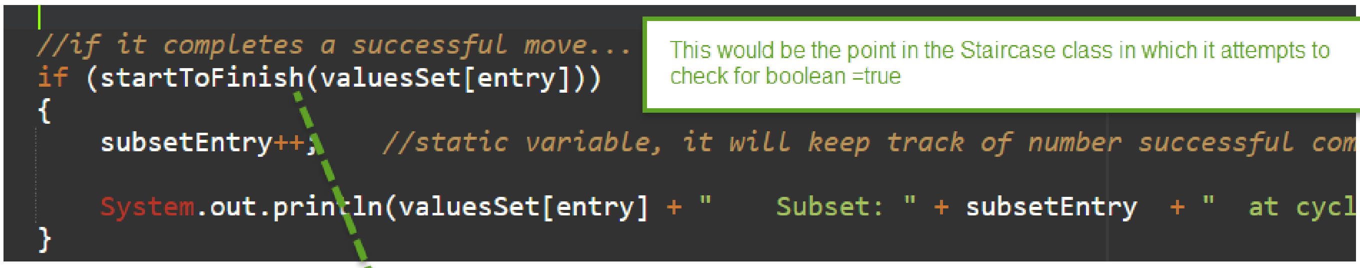
This is a reminder of the issue:



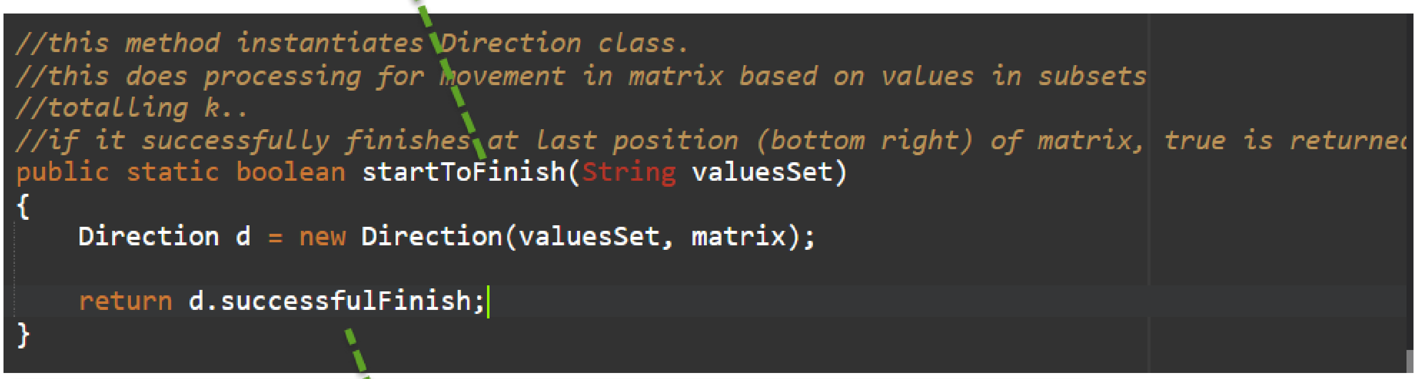
```
//this now processes each String in the ValuesSet (which will be Strings not outputted y
for (int entry=0; entry<valuesSet.length; entry++)
{
    if (valuesSet[entry]!="ALREADY PROCESSED")    //as per above, it needs bypass these
    {
```



```
//if it completes a successful move...
if (startToFinish(valuesSet[entry]))
{
    subsetEntry++;    //static variable, it will keep track of number successful com

    System.out.println(valuesSet[entry] + "    Subset: " + subsetEntry  + "  at cycl
}
```

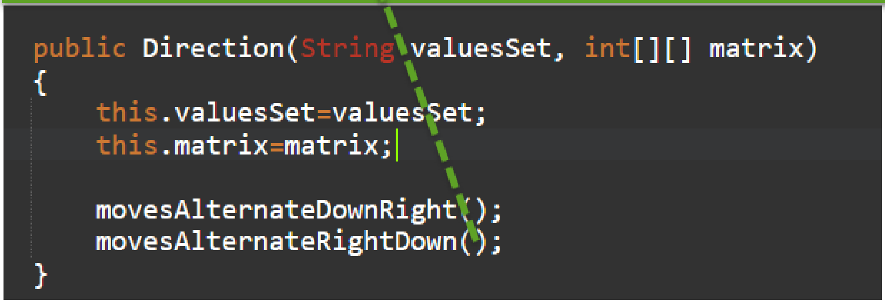
This would be the point in the Staircase class in which it attempts to check for boolean =true

This would invoke the startToFinish method and it would create a new instance of the Direction class



```
//this method instantiates Direction class.
//this does processing for movement in matrix based on values in subsets
//totalling k..
//if it successfully finishes at last position (bottom right) of matrix, true is returnec
public static boolean startToFinish(String valuesSet)
{
    Direction d = new Direction(valuesSet, matrix);

    return d.successfulFinish;
}
```

In the constructor Direction it would call both scenarios...



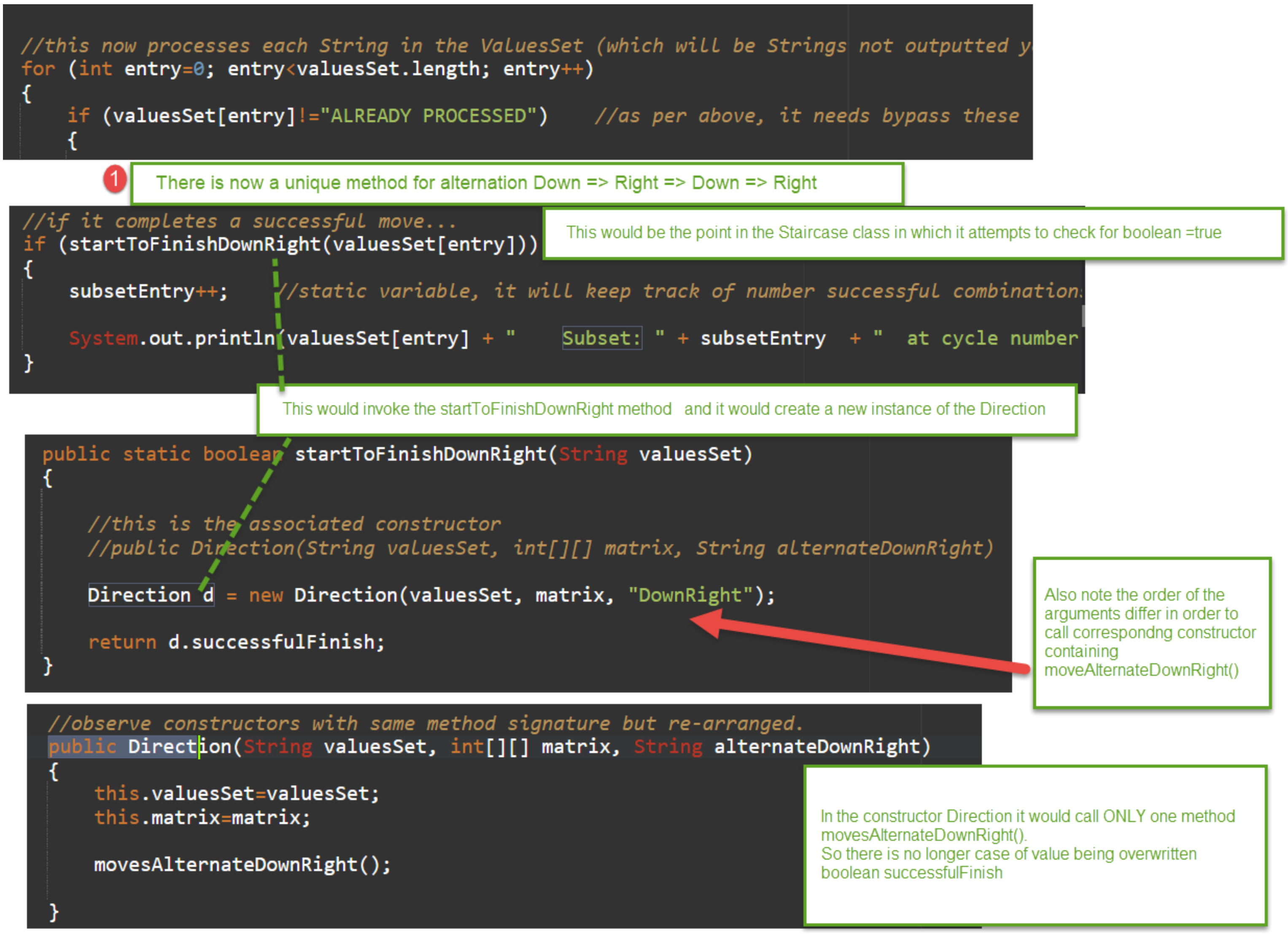
```
public Direction(String valuesSet, int[][] matrix)
{
    this.valuesSet=valuesSet;
    this.matrix=matrix;

    movesAlternateDownRight();
    movesAlternateRightDown();
}
```

Both set the boolean for successfulFinish and it will be overwritten. So, the value in method will always be from the last method...

One possible solution has occurred. And unfortunately, since it was not a technique not used in my coding before, it has taken a while to realise this. Constructor overloading.
I can then separate out the two methods into separate constructors.

Here is how I managed to perform the operation, it required lots of small changes.

But fortunately it functioned, which means I can perform this challenge knowing that end user will get some results in event that available memory elapses...



```
//this now processes each String in the ValuesSet (which will be Strings not outputted y
for (int entry=0; entry<valuesSet.length; entry++)
{
    if (valuesSet[entry]!="ALREADY PROCESSED")    //as per above, it needs bypass these
    {
```

1 There is now a unique method for alternation Down => Right => Down => Right

```
//if it completes a successful move...
if (startToFinishDownRight(valuesSet[entry]))
{
    subsetEntry++;    //static variable, it will keep track of number successful combination
    
    System.out.println(valuesSet[entry] + "    Subset: " + subsetEntry  + "  at cycle number
}
```

This would be the point in the Staircase class in which it attempts to check for boolean =true

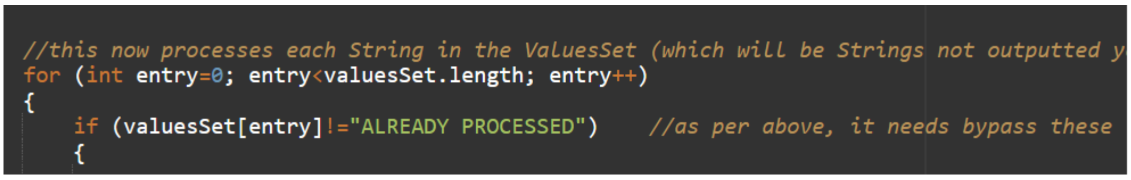
This would invoke the startToFinishDownRight method and it would create a new instance of the Direction

```
public static boolean startToFinishDownRight(String valuesSet)
{
    
    //this is the associated constructor
    //public Direction(String valuesSet, int[][] matrix, String alternateDownRight)
    
    Direction d = new Direction(valuesSet, matrix, "DownRight");
    
    return d.successfulFinish;
}
```

Also note the order of the arguments differ in order to call correspondng constructor containing moveAlternateDownRight()

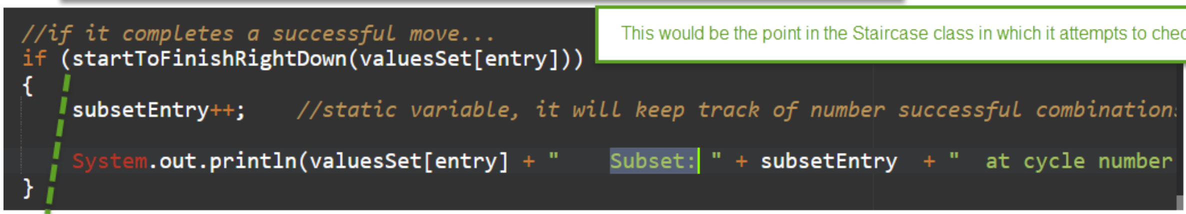
```
//observe constructors with same method signature but re-arranged.
public Direction(String valuesSet, int[][] matrix, String alternateDownRight)
{
    this.valuesSet=valuesSet;
    this.matrix=matrix;
    
    movesAlternateDownRight();
    
}
```

In the constructor Direction it would call ONLY one method movesAlternateDownRight().
So there is no longer case of value being overwritten boolean successfulFinish



```
//this now processes each String in the ValuesSet (which will be Strings not outputted y
for (int entry=0; entry<valuesSet.length; entry++)
{
    if (valuesSet[entry]!="ALREADY PROCESSED")    //as per above, it needs bypass these
    {
```

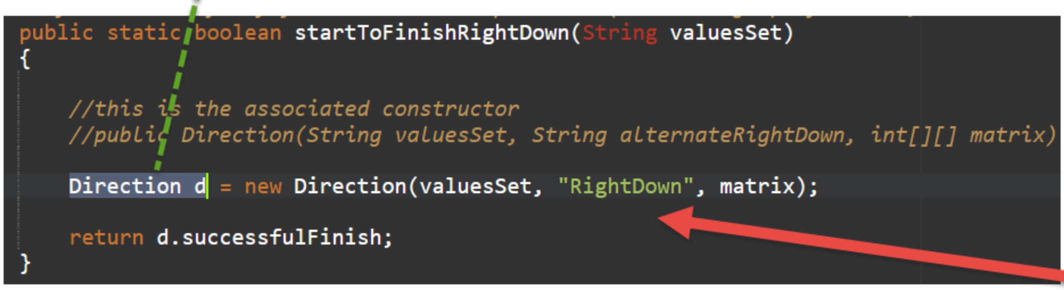
1 There is now a unique method for alternation Right => Down => Right => Down



```
//if it completes a successful move...
if (startToFinishRightDown(valuesSet[entry]))
{
    subsetEntry++;    //static variable, it will keep track of number successful combination

    System.out.println(valuesSet[entry] + "     Subset: " + subsetEntry   + "  at cycle number
}
```

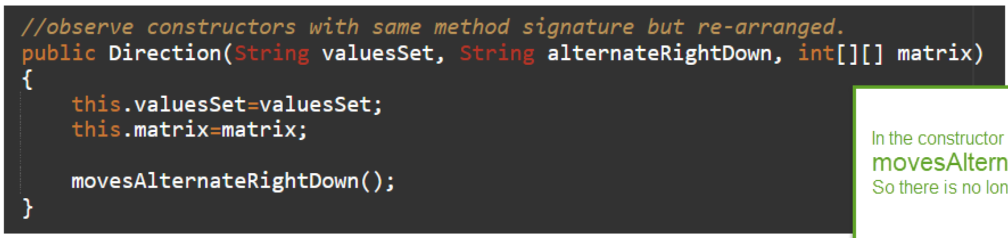
This would be the point in the Staircase class in which it attempts to check for boolean =true

This would invoke the startToFinishRightDown method   and it would create a new instance of the Direction



```
public static boolean startToFinishRightDown(String valuesSet)
{

    //this is the associated constructor
    //public Direction(String valuesSet, String alternateRightDown, int[][] matrix)

    Direction d = new Direction(valuesSet, "RightDown", matrix);

    return d.successfulFinish;
}
```

Also note the order of the arguments differ in order to call correspondng constructor containing moveAlternateRightDown()



```
//observe constructors with same method signature but re-arranged.
public Direction(String valuesSet, String alternateRightDown, int[][] matrix)
{
    this.valuesSet=valuesSet;
    this.matrix=matrix;

    movesAlternateRightDown();
}
```

In the constructor Direction it would call ONLY one method movesAlternateRightDown().
So there is no longer case of value being overwritten

Now, I get correct outputs at end execution:

```
***********SOLUTIONS************
2,1,6,2,1,1    Subset: 1  Alternating RIGHT => DOWN => RIGHT.......
5,1,3,2,1,1    Subset: 2  Alternating RIGHT => DOWN => RIGHT.......
1,1,7,2,1,1    Subset: 3  Alternating RIGHT => DOWN => RIGHT.......
3,1,5,2,1,1    Subset: 4  Alternating RIGHT => DOWN => RIGHT.......
4,1,4,2,1,1    Subset: 5  Alternating RIGHT => DOWN => RIGHT.......
```

I also get the following identical outcomes during its main execution:

```
2,1,6,2,1,1   (Alternating RIGHT => DOWN => RIGHT.......)   Subset: 1  at cycle number: 5005000
5,1,3,2,1,1   (Alternating RIGHT => DOWN => RIGHT.......)   Subset: 2  at cycle number: 5005000
1,1,7,2,1,1   (Alternating RIGHT => DOWN => RIGHT.......)   Subset: 3  at cycle number: 5005000
3,1,5,2,1,1   (Alternating RIGHT => DOWN => RIGHT.......)   Subset: 4  at cycle number: 5005000
4,1,4,2,1,1   (Alternating RIGHT => DOWN => RIGHT.......)   Subset: 5  at cycle number: 5005000
```