**** INITIAL NOTES **********

Can try recursively (going to tackle in a similar layout as CodingBat)...
use a single method to drive logic recursively.. it will present more readily as recursive exercise...

Can use %10 to get the last digit.
Place this in an array.
Can use % 10 again to get last digit.

To create the ascending number
Check array from right hand side, place it at end if larger

To create the descending number
check array from left hand side, place it at front if smaller number

The value that gets passed into recursive call is (number = number/10)
This will give the number excluding the large digit.
Do not believe any other variables required in recursive call.

We know reached the last digit once this value is equal to 0.....

EXTRA
As per of the validation rules, can initially run through the number inputted.
Can readily check if 4 digits by completing num/10 , should reach 4 and then become 0
In regards to 2 distinct numbers... , can process num%10 and check value against an integer array {0,1,2,3,4,5,6,7,8,9}

********** OUTPUT **********

No attempts have been made for validation to keep the simplicity flowing:

```
static int N=1234;
```

```
Welcome to Online IDE!! Happy Coding :)
4321-1234=3087
8730-378=8352
8532-2358=6174
Number executions:3
```

********** OUTPUT **********

```
static int N=0001;
```

It can be seen that in the process of

```
Welcome to Online IDE!! Happy Coding :)
1000-1=999
9990-999=8991
9981-1899=8082
8820-288=8532
8352-2538=5814
8451-1548=6903
9360-639=8721
8271-1728=6543
6453-3546=2907
9720-279=9441
9441-1449=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-378=8352
8532-2358=6174
Number executions:16
```

I have changed the code slightly so that the screen output of subtraction now shows the StringJoiner as oppose to the Integer.ValueOf StringJoiner

Previous code:

```
//Only significant screen output showing the subtraction...
System.out.println(Integer.valueOf(sj1.toString()) + "-" + Integer.valueOf(sj.toString()) + "=" + subtraction);
```

New code:

```
//Only significant screen output showing the subtraction...
System.out.println((sj1.toString() + "-" + (sj.toString() + "=" + subtraction)));
```

But it can be seen that it has still left me with an issue in subtraction value since it has taking off the leading 0 with integer...

//when N=1011 or N=1110 or N=0111 or N=1101 first outcome is 1110-0111=999
//when N=1100 or N=1010 or N=0011 or N=0110 first outcome is 1010-**0101**=909
//when N=1000 or N=0100 or N=0010 or N=0001 first outcome is 1000-0001=999
I did not take screenshots, but I took copy and paste from the console. It can be seen it is erroneous the minimum in **bold**. For now, I am ignoring this and will test all scenarios with 0 and 1 slightly further in this document.

I do not believe Java has a system to preserve this 0. And in all circumstances it loses a single 0.

Only option was to create an if else statement which examines if the number digits in the total is less than 4. It would then append an extra leading 0.

```
subtraction = Math.subtractExact(Integer.valueOf(sj1.toString()), Integer.valueOf(sj.toString()));

//keeps a string copy since in numbers such as 0001 or 0011 or 0111 it would drop off
//leading 0's
subtractionResult=Integer.toString(subtraction);

//this would only occur if  digit 1 appears with digit 0 in the four digit N
if (subtractionResult.length()<4)
{
    System.out.println((sj1.toString() + "-" + (sj.toString() + "=" + "0"+subtractionResult)));
}
else
{
    //Only significant screen output showing the subtraction...
     System.out.println((sj1.toString() + "-" + (sj.toString() + "=" + subtraction)));
}
```

The following output is now correct:

```
Welcome to Online IDE!! Happy Coding :)
1000-0001=0999
9990-0999=8991
9981-1899=8082
8820-0288=8532
8352-2538=5814
8451-1548=6903
9360-0639=8721
8271-1728=6543
6453-3546=2907
9720-0279=9441
9441-1449=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:16
```

*** INVESTIGATING THIS ANAMOLY **********

//when N=1100 or N=1010 or N=0011 or N=0110 first outcome is 1010-**0101**=909

At this point, I am just unsure if it is human error or whether it is something computationally not correct. As suggested above, I am trying N for all possibilities of 0 and 1

*** TEST CASE ***

```
static int N=0001;
```

This appears correct

```
Welcome to Online IDE!! Happy Coding :)
1000-0001=0999
9990-0999=8991
9981-1899=8082
8820-0288=8532
8352-2538=5814
8451-1548=6903
9360-0639=8721
8271-1728=6543
6453-3546=2907
9720-0279=9441
9441-1449=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:16
```

*** TEST CASE ***

```
static int N=0010;
```

Totally incorrect starting point

```
Welcome to Online IDE!! Happy Coding :)
8000-0008=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:6
```

*** TEST CASE ***

```
static int N=0011;
```

Totally incorrect starting point

```
Welcome to Online IDE!! Happy Coding :)
9000-0009=8991
9981-1899=8082
8820-0288=8532
8352-2538=5814
8451-1548=6903
9360-0639=8721
8271-1728=6543
6453-3546=2907
9720-0279=9441
9441-1449=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:15
```

*** TEST CASE ***

```
static int N=0100;
```

Totally incorrect starting point

```
Welcome to Online IDE!! Happy Coding :)
6400-0046=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:4
```

*** TEST CASE ***

```
static int N=0101;
```

Totally incorrect starting point

```
Welcome to Online IDE!! Happy Coding :)
6500-0056=6444
6444-4446=1998
9981-1899=8082
8820-0288=8532
8352-2538=5814
8451-1548=6903
9360-0639=8721
8271-1728=6543
6453-3546=2907
9720-0279=9441
9441-1449=7992
9972-2799=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:16
```

*** TEST CASE ***

```
static int N=0110;
```

Totally incorrect starting point

```
Welcome to Online IDE!! Happy Coding :)
7200-0027=7173
7731-1377=6354
6543-3456=3087
8730-0378=8352
8532-2358=6174
Number executions:5
```

*** TEST CASE ***

```
static int N=0111;
```

```
Welcome to Online IDE!! Happy Coding :)
7300-0037=7263
7632-2367=5265
6552-2556=3996
9963-3699=6264
6642-2466=4176
7641-1467=6174
Number executions:6
```

*** TEST CASES ***

No issues found:

N= 1000, 1001, 1010, 1011, 1100, 1101, 1110

So it seems there are issues with leading 0's in N.
And its creating an unexpected ascending number