Performing operation on 121

It would be straight forward analysing number via:
number%10
121%10 = 1 since (12 x 10) = 120
121 -120 =1
It would store lastDigit =1 (number%10)

If recursion is to be utilized, which appears to be strategic approach:
Given that above number is integer, no efforts will be used to perform autoboxing such as to Integer.....
It will make no difference whatsoever if initialised as primitive or autoboxed, since there will be no length method available on either approach.

So using previous techniques, it will continue performing:

(int)121/10 = 12
(int) 12/10 = 1
(int) 1/10 = 0 (this will not be included in the counter for divideBy10Required).
Now, it will store the firstDigit =1
It will compare the firstDigit against lastDigit. Both are equal.
divideBy10Required = 2 (number times division occurred to reach first digit).

isPalindrome will be set to true.

It will now perform recursion again
(**instead it will pass Number%10 into the call (12), having stripped off last digit**).
It will perform divideBy10Required % 2. If it is 0, it can halve the divideBy10Required.
This will enable it to perform passed value /10 (half amount of times = 1).
We know that if divideBy10Required =1, the passed integer value is two digits wide.
No comparisons are required since it is ascertained it is palindrome already reaching at this phase.... 121 (2 has no impact)

if divideBy10Required % 2 !=0, the value will not be able to be halved..
This might occur on a scenario such as 162341 (divideBy10Required= 5).
In this instance, it would have performed following recursion call on second attempt with value 16234
It would just perform similar operation lastDigit =4 and until divideBy10Required= 3

1623 will be now passed into recursive call
lastDigit = this would be derived by performing number % 10
NOTE: Since one digit is stripped off the end and firstDigit has moved inwards by

1, there will always be a deduction by two in variable divideBy10Required.

So in this case, divideBy10Required = 1.
It has to wind down the comparison slightly differently...

In order to obtain this lastDigit, following is required (usual process)
lastDigit = number % 10 = 3

firstDigit (penultimate digit) = 162**3**

To calculate firstDigit
1623 / 10 = 162
162 % 10 = 2
firstDigit = 2

NOTE: if the starting number was 66, divideBy10Required = 0 since its first execution.... So it will processed as usual and not affected by divideBy10Required
lastDigit = 66 % 10 = 6
recursive call with (int) 66/10 = 6
It will perform check if (6%10)== 0 (true)
firstDigit = 6 lastDigit = 6 (palindrome).

To ensure logic is correct for decrement of divideBy10Required by 2, the initial number is being extended and ensuring that divideBy10Required % 2 !=0
Also assume at each recursive call the number still qualifies even though on example it is not palindromic **(isPalindrome):**
Example number: 1345654631 divideBy10Required= 9
Passed into recursive call: 134565463
Digits of interest: 134565463
divideBy10Required is now equal to 7 to ensure firstDigit = 3
Note: firstDigit is obtained via number%10

(int) 134565463 / 10 = 13456546
13456546 / 10 = 1345654
1345654 / 10 = 134565
134565 / 10 = 13456
13456 / 10 = 1345
1345 / 10 = 134
134 / 10 = 13 **(7 executions).**
13

Let us assume that the remaining portion is still valid (isPalindrome):
Passed into recursive call: 13456546
Digits of interest: 13456546
divideBy10Required % 2 !=0 so:divideBy10Required is now equal to 5 (7-2) to ensure firstDigit = 4

Let us assume that the remaining portion is still valid (isPalindrome):
Passed into recursive call: 1345654
Digits of interest: 1345654
divideBy10Required is now equal to 3 (5-2) to ensure firstDigit = 5

Let us assume that the remaining portion is still valid (isPalindrome):
Passed into recursive call: 134565
Digits of interest: 134565
divideBy10Required is now equal to 1 (3-2) to ensure firstDigit = 6

It has to wind down the comparison slightly differently...
In order to obtain this lastDigit, following is required:
lastDigit = number % 10 = 5

firstDigit (penultimate digit) = 13456 5

To calculate firstDigit
134565 / 10 = 13456
13456 % 10 = 6
firstDigit = 6

ALONG THE ABOVE WHOLE PATH, divideBy10Required % 2 has always been !=0 (odd number).

But what if it results in an even number divideBy10Required % 2 ==0

Performing operation on 1245331

Passed into recursive call: 12451
Digits of interest: 12451
divideBy10Required is now equal to 4 to ensure firstDigit = 1 lastDigit = 1
Passed into recursive call: 1245
Digits of interest: 1245
divideBy10Required % 2 ==0 so:

divideBy10Required is now equal to 2 (4/2) to ensure firstDigit = 2 lastDigit = 5

Passed into recursive call: 124
Digits of interest: 124
divideBy10Required % 2 ==0 so:
divideBy10Required is now equal to 1 (2/1)
When divideBy10Required ==1 there is simply no more comparisons required.
The final state of isPalindrome will dictate the outcome.

But how is this scenario of **divideBy10Required ==1 (halving) differentiated from the divideBy10Required ==1 (decrementing by 2).**
It is required to be differentiated since the last process undertaken varies....
ONLY WAY TO HANDLE THIS IS IF IT SETS A BOOLEAN FLAG WHEN PERFORMING /10 (ODD NUMBER EXECUTIONS) OR EVEN NUMBER.
Please note both are mutually exclusive up to the point that divideBy10Required =1

Now, it will ensure that when it reaches divideBy10Required =1,

1) Make decision on palindrome status (ODD FLAG SET).
2) Continue final execution of /10 AND %10 (EVEN FLAG SET).

I believe I have now the correct logic to attempt a valid implementation.
Once again, it is a hybrid of some techniques applied in past.
But it most definitely required this roadmap, otherwise can envisage issues.