formula =
pSquareNum[i * (j * j)]
+ 1

+1 for representation
current perfect number
that used in current
calculation (j * j)

Loop condition: j = 1; j * j <= i; j++

n = 5

1. i = 1, j = 1
pSquareNum[1 - (1 * 1)] + 1
pSquareNum[0] + 1
0 + 1 = 1
minNumber = 1 (1^2)

2. i = 2, j = 1
pSqureNum[2 - (1 * 1)] + 1
pSquareNum[1] + 1
1 + 1 = 2
minNumber = 2 (1^2 + 1^2)

3. i = 3, j = 1
pSquareNum[3 - (1 * 1)] + 1
pSquareNum[2] + 1
2 + 1 = 3
minNumber = 3 (1^2 + 1^2 +1^2)

4. i = 4, j = 1
pSquareNum[4 - (1 * 1)] + 1
pSquareNum[3] + 1
3 + 1 = 4 (1^2 + 1^2 + 1^2 + 1^2)

from pSquareNum[3]
minNumber = 3

i = 4, j = 2
pSquareNum[4 - (2 * 2)] + 1
pSquareNum[0] + 1
0 + 1 = 1 (2^2)
Possibilities = 2
minNumber = 1

5. i = 5, j = 1
pSquareNum[5 - (1 * 1)] + 1
pSquareNum[4] + 1
1 + 1 = 2 (2^2 + 1^2)

from pSquareNum[4]
minNumber = 1

i = 5, j = 2
pSquareNum[5 - (2 * 2)] + 1
pSquareNum[1] + 1
1 + 1 = 2 (1^2 + 2^2)
Possibilities = 2
minNumber = 2

from pSquareNum[1]
minNumber = 1

Let's break down:

1. For **i** = 1, there's only one possibility: 1 itself. So, **pSquareNum[1]** = 1.
2. For **i** = 2, the possibilities are $\mathbf{1^2 + 1^2}$ = 2. The minimum number is 2 (1, 1), so **pSquareNum[2]** = 2.
3. For **i** = 3, the possibilities are $\mathbf{1^2 + 1^2 + 1^2}$ = 3. The minimum number is 3 (1, 1, 1), so **pSquareNum[3]** = 3.
4. For **i** = 4, the possibilities are $\mathbf{1^2 + 1^2 + 1^2 + 1^2}$ = 4, or $\mathbf{2^2}$ = 4. The minimum number is 1 (2 = 4 itself), so pSquareNum**[4]** = 1.
5. For **i** = 5, the possibilities are $\mathbf{2^2 + 1^2}$ = 5, or $\mathbf{1^2 + 2^2}$ = 5. The minimum number is 2 (1, 2), so **pSquareNum[5]** = 2.

And so on...

What actually happened here ?

n = 5

- 1^{st} loop for represents each number from 1 to n, which mean 1 to 5 and store each value that will be obtained in the second loop
- 2^{nd} loop is used to find the minimum number of perfect square numbers needed to reach the current i. And reuse the previously computed minimum number for the remaining value
- + 1 for represents the current perfect square used in the calculation.

  In this case, we will get 2 possibilities because of loop condition

  1. i = 5, j = 1

  pSquareNum[5 – (1 * 1)] + 1
  pSquareNum[4] + 1

  at this point we will directly reuse the previously computed minimum number at pSquareNum[4], which mean 1

  1 + 1 = 2
  1 (without current square) + 1 (current square)
  Minimum number at possibility 1 = 2

  2. i = 5, j = 2

  pSquareNum[5 – (2 * 2)] + 1

  pSquareNum[1] + 1

  at this point we will directly reuse the previously computed minimum number at pSquareNum[1], which mean 1

  1 + 1 = 2

1 (without current square) + 1 (current square)
Minimum number at possibility 2 = 2

Because both of possibilities minimum number for this case is 2.
And the result will be 2

```
pSquareNum = {int[13]@709} [0, 1, 2, 3, 1, 2, 3, 4, 2, 1, 2, 3, 3]
    0 = 0
    1 = 1
    2 = 2
    3 = 3
    4 = 1
    5 = 2
    6 = 3
    7 = 4
    8 = 2
    9 = 1
    10 = 2
    11 = 3
    12 = 3
```

```java
public static int numSquares(int n) {
    int[] pSquareNum = new int[n + 1];

    for (int i = 1; i <= n; i++) {
       pSquareNum[i] = i;

       for (int j = 1; j * j <= i; j++) {
          int remainingValue = i - (j * j);
          pSquareNum[i] = Math.min(pSquareNum[i], pSquareNum[remainingValue] + 1);
       }
    }
    return pSquareNum[n];
}
```

The explanation for the code:

1. An array **pSquareNum** is created, for store the minimum number of perfect square numbers
2. The 1st loop iterates from 1 to **n**, it's for representing each number from 1 to **n**
3. The 2nd loop iterates over possible perfect square numbers (**j * j**) where **j** is less than or equal to the current index **i**
4. For each **j**, it calculates the minimum number of perfect square numbers needed to reach the current index **i** by subtracting (**j * j**) from **i**

a. **i** - (**j** * **j**): This expression represents the remaining value after subtracting a perfect square (**j** * **j**) from the current index **i**. In other words, it calculates the difference between the current number **i** and a perfect square (**j** * **j**)
b. **pSquareNum**[**i** - (**j** * **j**)]: This part retrieves the minimum number of perfect square numbers needed to reach the remaining value calculated in step **a**. The idea is to reuse the previously computed minimum number for the remaining value
c. + 1: Since we are using a perfect square (**j** * **j**), we add 1 to the minimum number obtained in step **b**. This addition for represent the current perfect square used in the calculation.

5. After both loops, the function returns the value stored in **pSquareNum[n].** The final line **pSquareNum[i] = Math.min(pSquareNum[i], minNumber)**; ensures that the array **pSquareNum** at index **i** contains the minimum number of perfect squares needed to sum up to **i.**