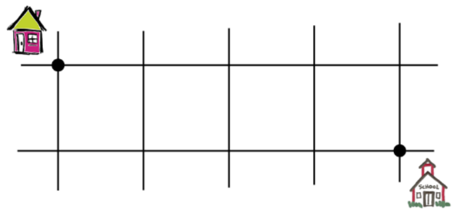
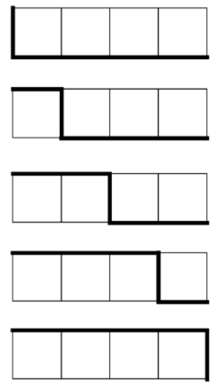
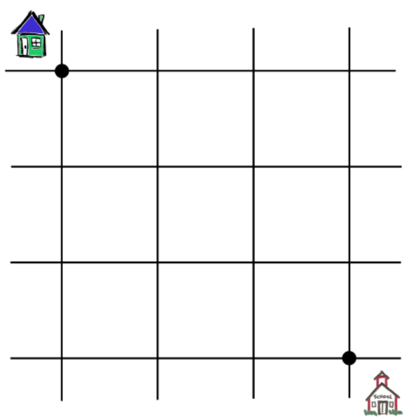
Assignment overview:

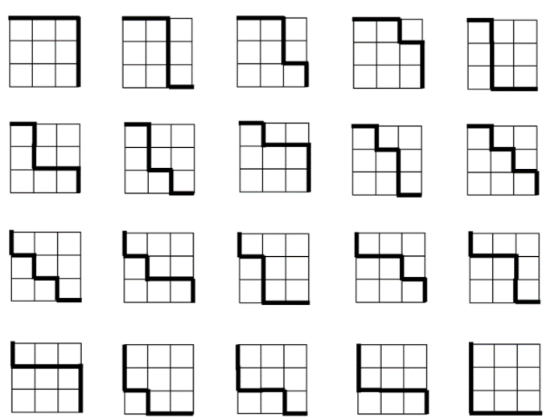
Young Johnny lives 5 blocks away from the schoolhouse as shown in the diagram below. When he walks to school he has a choice of five different routes he can take:





Young Alice lives just one block farther away from school (6 blocks), but surprisingly she can walk 20 different ways:





These numbers do not take into account the number of different times Johnny and Alice have to cross the street because of people not practicing proper physical distancing. Fortunately, your program doesn't need to worry about that, either.

Define *SW(m, n)* as the number of different ways that someone can walk *m* blocks down and *n* blocks over on a grid of city streets. There is no walking up or left. The starting point for a walk is (0, 0). Then *SW(1, 4)* = 5 is the number of ways Johnny can walk to school, and *SW(3, 3)* = 20 is the number of ways that Alice can walk to school.

Observe that for any points along the topmost (horizontal) street or the leftmost (vertical) street, there is only one way to walk there: a straight line from the starting point.

For all other grid points, in order to walk there, you must pass through either the point immediately above, or the point immediately to the left. Furthermore, these two sets of paths do not overlap, because one set uses a vertical street for the last segment, but the other uses a horizontal street last. Thus the number of different ways you can walk to any point on the grid is equal to the number of ways you could have arrived in the grid point immediately above PLUS the number of ways you could have arrived in the grid point immediately to the left.

Thus the function *SW(m, n)* can be calculated recursively as follows:

> *SW(0, n)* = 1 (There is only one way to walk straight to the right)
>
> *SW(m, 0)* = 1 (There is only one way to walk straight down)
>
> If *m, n* are both > 0: *SW(m,n) = SW(m-1, n) + SW(m, n-1)*

Part 1 (5 marks)

Write a function `SW_Recursive(m, n)` that calculates *SW(m, n)* recursively based on the definition given above.

In your `main()`, write a loop that uses `SW_Recursive(m, n)` to calculate *SW(i, i)* for successive values of i starting from 0 to at least 15. Use `System.currentTimeMillis()` to measure the execution time of your function, and report it along with the function's return value.

Here are the correct values for *SW(i, i)* up to *i* = 17:

```
SW_Recursive(0,0) = 1, time is 0 ms
SW_Recursive(1,1) = 2, time is 0 ms
SW_Recursive(2,2) = 6, time is 0 ms
SW_Recursive(3,3) = 20, time is 0 ms          ← There's Young Alice!
SW_Recursive(4,4) = 70, time is 0 ms
SW_Recursive(5,5) = 252, time is 0 ms
SW_Recursive(6,6) = 924, time is 0 ms
SW_Recursive(7,7) = 3432, time is 0 ms
SW_Recursive(8,8) = 12870, time is 0 ms
SW_Recursive(9,9) = 48620, time is 1 ms
SW_Recursive(10,10) = 184756, time is 1 ms
SW_Recursive(11,11) = 705432, time is 3 ms
SW_Recursive(12,12) = 2704156, time is 12 ms
SW_Recursive(13,13) = 10400600, time is 35 ms
SW_Recursive(14,14) = 40116600, time is 136 ms
SW_Recursive(15,15) = 155117520, time is 485 ms
SW_Recursive(16,16) = 601080390, time is 2094 ms
SW_Recursive(17,17) = 2333606220, time is 8118 ms
```

Part 2 (10 marks)

Write a second function `SW_DynamicProg(m, n)` that also calculates *SW(m, n)*. This function will use Dynamic Programming, but *not* recursion.

In your `main()`, write a loop that calculates `SW_DynamicProg(i, i)` for successive values of i from 0 to at least 30. Use `System.currentTimeMillis()` to measure the execution time of your function, and report it along with the function's return value.

Here are some more numbers for you to check your output. Note also that these calculations are *FAST* compared to the recursive version:

```
...
SW_DynamicProg(15,15) = 155117520, time is 0 ms
SW_DynamicProg(16,16) = 601080390, time is 0 ms
SW_DynamicProg(17,17) = 2333606220, time is 0 ms
SW_DynamicProg(18,18) = 9075135300, time is 0 ms
SW_DynamicProg(19,19) = 35345263800, time is 0 ms
SW_DynamicProg(20,20) = 137846528820, time is 0 ms
SW_DynamicProg(21,21) = 538257874440, time is 0 ms
SW_DynamicProg(22,22) = 2104098963720, time is 0 ms
SW_DynamicProg(23,23) = 8233430727600, time is 0 ms
SW_DynamicProg(24,24) = 32247603683100, time is 1 ms
SW_DynamicProg(25,25) = 126410606437752, time is 0 ms
SW_DynamicProg(26,26) = 495918532948104, time is 0 ms
SW_DynamicProg(27,27) = 1946939425648112, time is 0 ms
SW_DynamicProg(28,28) = 7648690600760440, time is 0 ms
SW_DynamicProg(29,29) = 30067266499541040, time is 0 ms
SW_DynamicProg(30,30) = 118264581564861424, time is 0 ms
```

How far can you go?

PFG

One (1) PFG if your program can (correctly) calculate *SW(37, 37)*.

Submission information

Due date: As shown on Learning Hub. Late assignments will not be graded.

What to submit: Just your java code

You may (and should!) discuss the lab and coding techniques with your classmates, but all of the code you submit to Learning Hub must be your own.

This assignment is worth 15 marks.