LOGIC

Rounding In Millions

Published by Deep Xavier in Java

language_fundamentals | loops | math | numbers | objects

Given an array of cities and populations, return an array where all populations are **rounded to the nearest million**.

Examples

```
millionsRounding([
  ["Nice", 942208],
  ["Abu Dhabi", 1482816],
  ["Naples", 2186853],
  ["Vatican City", 572]
]) → [
  ["Nice", 1000000],
  ["Abu Dhabi", 1000000],
  ["Naples", 2000000],
  ["Vatican City", 0]
]
```

```
millionsRounding([
  ["Manila", 13923452],
  ["Kuala Lumpur", 7996830],
  ["Jakarta", 10770487]
]) → [
  ["Manila", 14000000],
  ["Kuala Lumpur", 8000000],
  ["Jakarta", 11000000]
]
```

Notes

Round down to **0** if a population is below **500,000.**

There is tendency to jump into the coding and having to face unprecedented improvisations. I will spend a bit more effort to try and understand issue first.

I will populate the information into an Object array / alternatively String 2d array.
Process conversion of the number(long) into Object or String respectively.

Firstly, my previous exercises, I have used math.round and math.ceil to round integer upwards. This is not a requirement for this exercise since population are whole figures.

Will start with 780,000, We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 1. It can be assigned to variable frequencyRoundedNumber.

We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
largePortion = frequencyRoundedNumber x 500,000 = 500,000
smallPortion = 780,000 – (1 x 500,000) = 280,000
Now smallPortion can be analysed by method call to roundSmallPortion
It performs following checks in the method.
checkNegativeNumber = smallPortion – 1,000,000 = 280,000 – 1,000,000 = -720,000
if checkNegativeNumber< -500,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;

if checkNegativeNumber>= -500,000 set smallPortion = 500,000
formattedRoundedNumber = largePortion + smallPortion;
= 500,000 + 500,000 = 1,000,000

Will start with number=3,274,214 We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 6. It can be assigned to variable frequencyRoundedNumber
We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
largePortion = frequencyRoundedNumber x 500,000 = 3,000,000
smallPortion = 3,274,214 – (6 x 500,000) = 274,214
Now smallPortion can be analysed by method call to roundSmallPortion.
It performs following calculation in the method:
checkNegativeNumber = smallPortion – 1,000,000 = 274,214 – 1,000,000 = -725,786
if checkNegativeNumber< -500,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;
=3,000,000 + 0 = 3,000,000;
if checkNegativeNumber>= -500,000 set smallPortionRounded=1,000,000
formattedRoundedNumber = largePortion + smallPortionRounded

method roundSmallPortion (return long)

IF SECTION
I can perform number%500 000. If the figure is 0, it creates few possibilities:
The number is exactly divisible by 500,000, so it can either be exactly multiple of million or multiple of

500,000. In layman terms, the solution is simply the number unless if frequencyRoundedNumber % 2 == 1 (this would be applicable for cases such as 500,000, 1,500,000, 2,500,000 . In all these cases, it can be seen that an additional 500,000 will be added to the number.. This will be the formattedRoundedNumber = number + 500,000.

return formattedRoundedNumber

Or the number can be 0 itself (0%500 000==0). This is not a valid test, since it can refer to any multiple of 500,000.

ALL of these are valid for if statements
frequencyRoundedNumber==0 (suggests number less than 500,000)
Or largePortion = 0 (frequencyRoundedNumber x 500,000)

Or number =0 (perhaps the most easiest)

In which case, the solution is 0
formattedNumber = 0;

return formattedRoundedNumber

END IF

ELSE SECTION:

if checkNegativeNumber< -500,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;

return formattedRoundedNumber

if checkNegativeNumber>= -500,000 set smallPortionRounded=1,000,000
formattedRoundedNumber = largePortion + smallPortionRounded

return formattedRoundedNumber

END ELSE

So this is the most comprehensive logic I have drafted since I envisage an almost exact translation to the above in the code.
The only scenario not tested is a number under 500,000. We know that it can be identified immediately, but I tried to maintain all logic as tidy as possible....

Will start with number=274,214 We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 274,214. It can be assigned to variable frequencyRoundedNumber
We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
We can see this will cause miscalculation of 274,214 – (274,214 x 500,000)
largePortion = frequencyRoundedNumber(274,214) x 500,000 = 137,107,000,000
I do not want to introduce logic for no apparent reason in the code, but it can be seen that this largePortion exceeds the initial number. But it is strictly speaking a valid number for a population.
Rather than focussing on a valid positive number, we know definitely that largePortion or smallPortion can not be negative, notably for a population..

So an entry will be needed in code:

if (smallPortion < 0), set smallPortion = number;
set largePortion = 0
set frequencyRoundedNumber = 0
Now smallPortion can be analysed by method call to roundSmallPortion.
I will just go through each area of the code bit by bit to identify if any different conditions will be triggered.

It will perform number%500 000 and this will be equal to number = 274,214 (not 0)
We know it is possible for number%500,000 to be equal exactly to frequencyRoundedNumber (for instance if number itself was genuinely over 500,000 and divisible). For instance a number of 2,500,005 could be as such since 500,000 x frequencyRoundedNumber**(5)** = 2,500,000 and also there is a remainder of 5 (2,500,005%500,000). Something I need to be aware of and my code is not touching on this logic.....
It appears it will enter else block.
Again, strictly speaking, I could have set logic so that if number<500,000, return formattedNumber...
But it is good practice to let this number also pass similarly to a number greater than 500,000.

It performs following calculation in the method:
checkNegativeNumber = smallPortion – 1,000,000 = 274,214 – 1,000,000 = -725,786
if checkNegativeNumber< -500,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;

= 0 (largePortion has been set as 0 since smallPortion is less than 0. Although this sounds counterintuitive, it perfectly logical if this section is analysed above) + 0 = 0;

return formattedRoundedNumber

if checkNegativeNumber>= -250,000 set smallPortionRounded=1,000,000
formattedRoundedNumber = largePortion + smallPortionRounded

return formattedRoundedNumber

I will simulate two more scenario before I complete the code. I will force it through the entire code for the below reasons:
I will include a potential red herring such as number = 500,001 due to getting number%500,000 = 1
and also frequencyRoundedNumber = 1 frequencyRoundedNumber%2==1
It is very important integrity of my validations are not compromised..

I will also include another such as number = 1 due to getting
number%500,000 = 1
frequencyRoundedNumber = 0
frequencyRoundedNumber%2==0

Will start with number=500,001 We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 1. It can be assigned to variable frequencyRoundedNumber
We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
= 500,001 - (1 x 500,000) = 1
largePortion = frequencyRoundedNumber(1) x 500,000 = 500,000

It will bypass this condition:
if (smallPortion < 0)

Now smallPortion can be analysed by method call to roundSmallPortion.
I will just go through each area of the code bit by bit to identify if any different conditions will be triggered.

It will perform number%500,000 and this will be equal to 1
We know it is possible for number%500,000 to be equal exactly to the frequencyRoundedNumber. As explained above... I am checking the if statement in the method if this causes conflict...
The only time it could cause conflict if both are set to 0...
So, I would need a simulation for this just to be sure.

It appears it will enter else block.
It performs following calculation in the method:
checkNegativeNumber = smallPortion – 1,000,000 = 1 – 1,000,000 = -999,999
if checkNegativeNumber< -250,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;
= 500,000 + 1 = 500,001;

return formattedRoundedNumber

if checkNegativeNumber>= -250,000 set smallPortionRounded=1,000,000
formattedRoundedNumber = largePortion + smallPortionRounded

return formattedRoundedNumber

Will start with number=1 We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 0. It can be assigned to variable frequencyRoundedNumber
We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
= 1 - (0 x 500,000) = 1
largePortion = frequencyRoundedNumber(0) x 500,000 = 0

It will bypass this condition:
if (smallPortion < 0)

Now smallPortion can be analysed by method call to roundSmallPortion.
I will just go through each area of the code bit by bit to identify if any different conditions will be triggered.

It will perform number%500,000 and this will be equal to 1
We know it is possible for number%500,000 to be equal 1 and frequencyRoundedNumber as 0. As explained above... I am checking the if statement in the method if this causes conflict...
Again, both are not set to 0.

It appears it will enter else block.
It performs following calculation in the method:
checkNegativeNumber = smallPortion – 1,000,000 = 1 – 1,000,000 = -999,999

if checkNegativeNumber< -250,000 set smallPortionRounded = 0
formattedRoundedNumber = largePortion + smallPortionRounded;
= 0 + 1 = 1;

return formattedRoundedNumber

if checkNegativeNumber>= -250,000 set smallPortionRounded=1,000,000
formattedRoundedNumber = largePortion + smallPortionRounded

return formattedRoundedNumber

This will be final simulation:
Will start with number=0 We know a strategic approach would be to dissect this number so that one portion conforms to large part (multiple of 500,000) and other small part.
So, simplest technique would be to perform (int) number/500,000.
This would return 0. It can be assigned to variable frequencyRoundedNumber
We can let the smallPortion = number – (frequencyRoundedNumber x 500,000)
= 0 - (0 x 500,000) = 0
largePortion = frequencyRoundedNumber(0) x 500,000 = 0

It will bypass this condition:
if (smallPortion < 0)

Now smallPortion can be analysed by method call to roundSmallPortion.
I will just go through each area of the code bit by bit to identify if any different conditions will be triggered.
Note, all variables above are 0. **This is only scenario I can validate a 0 population without explicitly interrogating the number variable…**
So it will return the formattedNumber = 0 + 0 = 0

return formattedRoundedNumber