I have finished the challenge and almost exactly translated the logic into the code.
I can immediately see that all test cases are not correct....
Here is the output of the example populations.. If I fail to identify the issue, I will perform my test cases for extensive troubleshooting. Anyhow, as always, my test cases will be exhaustive..

```
Welcome to Online IDE!! Happy Coding :)

Original population:

[Nice, 942208]

[Abu Dhabi, 1482816]

[Naples, 2186852]

[Vatican City, 572]


Population rounded to nearest 1000000

[Nice, 500000]

[Abu Dhabi, 1000000]

[Naples, 2000000]

[Vatican City, 0]
```

A good starting point will be to add a System output at each junction.. However whilst I checked code, I identified something immediately. I can see that roundingUnit should have been -roundingUnit

```
else
    {
       System.out.println("Population is not divisible by: " + roundingUnit);
       if (checkNegativeNumber<roundingUnit)
       {
```

I am completing this change and running the code again. It can now be seen that all the test cases are failing!

```
Population rounded to nearest 1000000

[Nice, 1500000]
```

[Abu Dhabi, 2000000]

[Naples, 3000000]

[Vatican City, 1000000]

I will continue adding useful screen outputs and check the code flow.
I immediately noticed I missed a critical line of code:
checkNegativeNumber = smallPortion – roundingToNearest;

Here is the output, I have highlighted the correct output at end....
Green = correct red = incorrect
It appears I am getting exactly the same output as my first attempt. Nice is still erroneous. It should be 1,000,000 however code is generating 500,000

Firstly, it can be seen that in all examples, the largePortion is either 0 or a multiple of roundingToNearest (1,000,000). And smallPortion is less than roundingUnit(500,000). This will inevitably render all populations to round downwards..

Nice is the exception with largePortion = 500,000

It has immediately occurred straight away that I have not included logic in my code to evaluate this. But I am not entirely sure where to implement this...
However we know that largePortion is divisible by 500,000 and gives an odd number...
For instance 500,000/500,000= 1 1,500,000/500,000 = 3
So can configure frequencyRoundedNumberInLargePortion = largePortion/roundingUnit;

If ((frequencyRoundedNumberInLargePortion%2) == 1)
//we then know we are dealing with largePortion such as 500,000, 1,500,000, 2,500,000.
so formattedRoundedNumber = largePortion + roundingUnit

It has also expanded the scope of else section of the if-else

**********INCORRECT OUTPUT**********

Welcome to Online IDE!! Happy Coding :)

Original population:

[Nice, 942208]

[Abu Dhabi, 1482816]

[Naples, 2186852]

[Vatican City, 572]

Processing population for: Nice

Processing the small portion: 442208

This is large portion: 500000

Checking negative number: -557792

Population is not divisible by: 500000

Small portion is less than: 500000

Processing population for: Abu Dhabi

Processing the small portion: 482816

This is large portion: 1000000

Checking negative number: -517184

Population is not divisible by: 500000

Small portion is less than: 500000

Processing population for: Naples

Processing the small portion: 186852

This is large portion: 2000000

Checking negative number: -813148

Population is not divisible by: 500000

Small portion is less than: 500000

Processing population for: Vatican City

Processing the small portion: 572

This is large portion: 0

Checking negative number: -999428

Population is not divisible by: 500000

Small portion is less than: 500000

Population rounded to nearest 1000000

[Nice, 500000]

[Abu Dhabi, 1000000]

[Naples, 2000000]

[Vatican City, 0]

*********NEW OUTPUT***************FUNCTIONAL

lcome to Online IDE!! Happy Coding :)

Original population:

[Nice, 942208]

[Abu Dhabi, 1482816]

[Naples, 2186852]

[Vatican City, 572]

Processing population for: Nice

Processing the small portion: 442208

This is large portion: 500000

Checking negative number: -557792

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 1

Large portion population is 500,000 , 1,500,000, 2,500,000 etc

Processing population for: Abu Dhabi

Processing the small portion: 482816

This is large portion: 1000000

Checking negative number: -517184

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 2

Small portion is less than: 500000

Processing population for: Naples

Processing the small portion: 186852

This is large portion: 2000000

Checking negative number: -813148

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 4

Small portion is less than: 500000

Processing population for: Vatican City

Processing the small portion: 572

This is large portion: 0

Checking negative number: -999428

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 0

Small portion is less than: 500000

Population rounded to nearest 1000000

[Nice, 1000000]

[Abu Dhabi, 1000000]

[Naples, 2000000]

[Vatican City, 0]

So this raises a few question about my design, since I deliberately coded to meet expectations of the challenge and not make it universal for all scenarios of different rounding presets.

If I had set the roundingToNearest to an odd number, we know straight away that roundingUnit will need some thought since roundToNearest can not be halved.
I just immediately set the roundingUnit to be half. I will add this small implementation into my code.

But assuming that roundingNearest was another even number such as 250,000.
roundingUnit = 125,000.

So similar to Nice [Nice, 942208] roundingUnit = 500000 roundingToNearest = 1000000

I will adjust the Nice population to be 235,522 (reduced by factor of 4) inline with:
Nice [Nice, 235522] roundingUnit = 250000 roundingToNearest = 125000

And I am hoping it still enters this part of the code.

```
System.out.println("What is frequencyRoundedNumberInLargePortion: " +
frequencyRoundedNumberInLargePortion);


        if(frequencyRoundedNumberInLargePortion%2==1)

        {

           System.out.println("Large portion population is in this sequence: " +
roundingUnit+ ", " + (roundingUnit+roundingToNearest)+ ", " +
(roundingUnit+roundingToNearest+roundingToNearest+" ....."));

           formattedRoundedNumber = largePortion + roundingUnit;

        }
```

And it does correctly enter:

```
Processing population for: Nice(235522)
Processing the small portion: 110522
This is large portion: 125000
Checking negative number: -139478
Population is not divisible by: 125000
What is frequencyRoundedNumberInLargePortion: 1
Large portion population is in this sequence: 125000, 375000, 625000 .....
Population: 235522=> 250000 (rounded to nearest 250000)
```

So if my understanding is correct of my code, if the population is now increased to over 250,000 and less than 375,000 (it will not enter this section).
500,000 and less than 625,000 (it will not enter this section).
.....
I will quickly try to experiment with code and perform rounding to 2,000,000.
No issues whatsoever.

So as a last test, I want to quickly understand impact of the code if set roundingUnit to (roundingToNearest + 1)/2 if the roundingtoNearest is 11

Nice [Nice, 22] roundingUnit = 6 roundingToNearest = 11

It can be seen this is invalid:

```
Processing population for: Nice(11)
Processing the small portion: 5
This is large portion: 6
Checking negative number: -6
Population is not divisible by: 6
What is frequencyRoundedNumberInLargePortion: 1
Large portion population is in this sequence: 6, 17, 28 .....
Population: 11=> 12 (rounded to nearest 11)
```

Nice [Nice, 20] roundingUnit = 6 roundingToNearest = 11
It can be seen this is invalid:

```
Processing population for: Nice(20)
Processing the small portion: 2
This is large portion: 18
Checking negative number: -9
Population is not divisible by: 6
What is frequencyRoundedNumberInLargePortion: 3
Large portion population is in this sequence: 6, 17, 28 .....
Population: 20=> 24 (rounded to nearest 11)
```

I am going to perform all key calculations manually and trace through my code:
frequencyRoundedNumber = population/roundingUnit; =20/6 = 3
largePortion = frequencyRoundedNumber * roundingUnit =3*6 =18
smallPortion = population - (frequencyRoundedNumber * roundingUnit) = 20 – (3*6)=2
checkNegativeNumber = smallPortion – roundingToNearest = 2 - 11 = -9
divisbleByRoundingUnit = population%roundingUnit = 20 / 6 = 3.33333
frequencyRoundedNumberInLargePortion = largePortion/roundingUnit = 18/6 = 3

All the values correspond to the output above.

So I will run through my code.

We can see straight away, that if roundingToNearest = 9 then roundingUnit = 5, then we it hits this part of the code, it will evaluate as true....

```
//as per logic, this can occur numerous reasons
if (divisbleByRoundingUnit==0)
{
    System.out.println("Population is exactly divisible by: " + roundingUni
    //here we know number is 500,000, 1,500,000,  2,500,000
    // this is roundingUnit(500,000) and then cumulative total of
    //roundingUnit(500,000) + roundingToNearest (1,000,000);
    formattedRoundedNumber=largePortion+roundingUnit;
}
```

So formattedRoundedNumber = 18+6 = 24

This would completely be wrong logic, so I would need to include Boolean so that it does not enter if roundingToNearest was increased by 1.

I have implemented this and didn't expect it to fix issue with Nice since:

divisbleByRoundingUnit = population%roundingUnit = 20 / 6 = 3.33333

The first real part of code that raises query is this so far:

frequencyRoundedNumberInLargePortion = largePortion/roundingUnit = 18/6 = 3

So it enters this section and formattedRoundedNumber = 18 + 6 =24

This is where it generates the output which is incorrect.

```
System.out.println("What is frequencyRoundedNumberInLargePortion: " + frequencyRou

    if(frequencyRoundedNumberInLargePortion%2==1)
    {
        System.out.println("Large portion population is in this sequence: " + roun
        formattedRoundedNumber = largePortion + roundingUnit;
    }
```

Again, it is relating roundingUnit to a modified roundingToNearest.

So I need to apply the Boolean here also on the loop.

I have run the code again, and get a different result:

```
Processing population for: Nice(20)
Processing the small portion: 2
This is large portion: 18
Checking negative number: -9
Population is not divisible by: 6
What is frequencyRoundedNumberInLargePortion: 3
Small portion is less than: 6
Population: 20=> 18 (rounded to nearest 11)
```

So I will continue navigating through my code:

It can be seen that the code enters here since

checkNegativeNumber = smallPortion – roundingToNearest = 2 - 11 = -9

roundingUnit *-1 = -6

```
//the small portion is less than 500,000, so rounds downwards...
if (checkNegativeNumber< (-1*roundingUnit))
{
    System.out.println("Small portion is less than: " + roundingUnit);
    smallPortionRounded = 0;
    formattedRoundedNumber =  largePortion + smallPortionRounded;
}
```

We can not place a Boolean on here, since the code is almost finished..

We do know that 20 (rounded to nearest 11) should be 22 and not 18 (currently).

So, the code has rounded the number down to largePortion = 18.

```
Processing population for: Nice(20)
Processing the small portion: 2
This is large portion: 18
Checking negative number: -9
Population is not divisible by: 6
What is frequencyRoundedNumberInLargePortion: 3
Small portion is less than: 6
Population: 20=> 18 (rounded to nearest 11)
```

Firstly it is worth examining where the round should occur....

Multiples of 11 and 11, 22,33

So if small portion was 5 (less than 6 as stated in screen output), we would expect the initial population to be 18+5=23

AND clearly, this is not condition in which it should round down (see below).

It would round down to 11 (small portion = 12,13,14,15,16)

It would round up to 22 (small portion = 21,20,19,18,17)

It would round down to 22 (small portion = 23,24,25,26,27)

We can see in the example overleaf that rounding down to effectively largePortion is correct... (since largePortion is a multiple of roundingUnit 3).

Whereas above, roundingUnit is 6 (only due to adjusting associated roundToNearest from 11 to 12).

We ned to change the calculation of roundingUnit since every variable depends on this.

I am setting roundingUnit = roundToNearest

This is the wrong approach as can be seen below (with some of the outputs).

```
Processing population for: Nice(20)
Processing the small portion: 9
This is large portion: 11
Checking negative number: -2
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 1
Small portion is greater than or equal to: 11
Small portion is greater than or equal to: 11
Potential error in calculations, small portion being rounded up to nearest 11
Population: 20=> 22 (rounded to nearest 11)
```

```
Processing population for: Vatican City(572)
Processing the small portion: 0
This is large portion: 572
Checking negative number: -11
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 52
Small portion is greater than or equal to: 11
Small portion is greater than or equal to: 11
Potential error in calculations, small portion being rounded up to nearest 11
Population: 572=> 583 (rounded to nearest 11)
```

```
Processing population for: Abu Dhabi(1482816)
Processing the small portion: 5
This is large portion: 1482811
Checking negative number: -6
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 134801
Small portion is greater than or equal to: 11
Small portion is greater than or equal to: 11
Potential error in calculations, small portion being rounded up to nearest 11
Population: 1482816=> 1482822 (rounded to nearest 11)
```

We can see Nice is actually correct, but the logic is clearly wrong.. So I will analyse all these:

Nice

In summary, where it enters area in which it defines incorrect calculation
(due to if checkNegativeNumber>= -1*roundingUnit) -6 > -11

I can now check to see how the checkNegativeNumber was influenced by my code change above. **NOTE: If it was modified, it would need to be separate case for odd roundingToNearest variable**

If there is no direct impact, I would look to introduce logic in section of code in this colour

checkNegativeNumber = smallPortion – roundingToNearest

We know:

frequencyRoundedNumber = population/roundingUnit = 20/11 =1

smallPortion = population - (frequencyRoundedNumber * roundingUnit) = 20 – (1*11)=9

So now, the statement in this section of code is not correct at all, so it can be removed.

```
else  //(if checkNegativeNumber>=  -500,000)
{
    System.out.println("Small portion is greater than or equal to: " + roundingUnit);
    System.out.println("Small portion is greater than or equal to: " + roundingUnit);
```

I can start to apply code here now for this scenario, going back to this..
We can see that 5 entries round down and 5 entries round up...
It would round down to largePortion=11 (small portion = 12,13,14,15,16)
It would round up to largePortion=22 (small portion = 21,20,19,18,17)

So can perform roundingToNearest -1
And check smallPortion<= (roundingToNearest -1)/2
If so, it would round downwards.. formattedNumber = largePortion
otherwise formattedNumber = largePortion + roundingToNearest

I believe this is the entire code to completion, so will output results below and check the outcomes:

```
Processing population for: Nice(20)
Processing the small portion: 9
This is large portion: 11
Checking negative number: -2
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 1
Population: 20=> 22 (rounded to nearest 11)
```

```
Processing population for: Abu Dhabi(1482816)
Processing the small portion: 5
This is large portion: 1482811
Checking negative number: -6
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 134801
Population: 1482816=> 1482811 (rounded to nearest 11)
```

```
Processing population for: Naples(2186852)
Processing the small portion: 8
This is large portion: 2186844
Checking negative number: -3
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 198804
Population: 2186852=> 2186855 (rounded to nearest 11)
```

```
Processing population for: Vatican City(572)
Processing the small portion: 0
This is large portion: 572
Checking negative number: -11
Population is not divisible by: 11
What is frequencyRoundedNumberInLargePortion: 52
Population: 572=> 572 (rounded to nearest 11)
```

TEST CASES : DUE TO NATURE OF THE CODE OUTPUT AS PER EXAMPLE, I AM PRESENTING ALL TEST CASES TOGETHER WHICH REFLECT EXAMPLES IN LOGICAL ANALYSIS.

Welcome to Online IDE!! Happy Coding :)

Original population:

[Nice, 274214]

[Abu Dhabi, 500001]

[Naples, 1]

[Vatican City, 0]

[London, 10234865]

[Birmingham, 1532896]

Processing population for: Nice(274214)

Processing the small portion: 274214

This is large portion: 0

Checking negative number: -725786

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 0

Small portion is less than: 500000

Population: 274214=> 0 (rounded to nearest 1000000)

Processing population for: Abu Dhabi(500001)

Processing the small portion: 1

This is large portion: 500000

Checking negative number: -999999

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 1

Large portion population is 500,000 , 1,500,000, 2,500,000 etc

Population: 500001=> 1000000 (rounded to nearest 1000000)

Processing population for: Naples(1)

Processing the small portion: 1

This is large portion: 0

Checking negative number: -999999

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 0

Small portion is less than: 500000

Population: 1=> 0 (rounded to nearest 1000000)

Processing population for: Vatican City(0)

Processing the small portion: 0

This is large portion: 0

Checking negative number: -1000000

Population is exactly divisible by: 500000

The initial population is 0:

Population: 0=> 0 (rounded to nearest 1000000)

Processing population for: London(10234865)

Processing the small portion: 234865

This is large portion: 10000000

Checking negative number: -765135

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 20

Small portion is less than: 500000

Population: 10234865=> 10000000 (rounded to nearest 1000000)

Processing population for: Birmingham(1532896)

Processing the small portion: 32896

This is large portion: 1500000

Checking negative number: -967104

Population is not divisible by: 500000

What is frequencyRoundedNumberInLargePortion: 3

Large portion population is 500,000 , 1,500,000, 2,500,000 etc

Population: 1532896=> 2000000 (rounded to nearest 1000000)

Population rounded to nearest 1000000

[Nice, 0]

[Abu Dhabi, 1000000]

[Naples, 0]

[Vatican City, 0]

[London, 10000000]

[Birmingham, 2000000]

** Process exited - Return Code: 0 **