Java challenge exercise #1 – Caching

This is an advanced and challenging exercise that really will challenge you as a developer!

1. The following class contains a very “complex” calculation that takes 5 seconds each time it is called.

```
public class Calculator {

    //Perform a very complex calulation that take 5 seconds to complete
    public int doComplexCalculation(int x) throws InterruptedException {

        System.out.println("Expensive calculation: Started for value x=" + x);
        Thread.sleep(1000);
        System.out.println("Expensive calculation: Done");

        return x*2;
    }
}
```

2. In our main method, we call this class with this code:

```
Calculator calc = new Calculator();
Random rand = new Random();

int maxValue = 10;

for (int i = 0; i < 1000; i++) {

    int value = rand.nextInt(maxValue);

    System.out.println("Getting result for value " + value);
    int result = calc.doComplexCalculation(value);

    System.out.println("Result: " + result);
    System.out.println();
}
```

   Run the code and make sure it works and that you understand it.

3. Customer feedback!

   The application works great! But they complain that the application is slow! So, what can we do?

   We know that the result of each calculation is always the same for a given input. What if we could remember and reuse the result instead of recalculate it each time?

4. Let's add a simple **caching layer** that will allow us to remember previous calculations. We put this logic in a separate class to follow the **single responsibility principle**.

   We basically put a class between the client and the calculator that will take care of remembering previous calculations.

5. The cache should be called using this code:

```
// Capacity is the number of unique items the cache can remember
int cacheCapacity = 10;
SimpleCache cache = new SimpleCache(cacheCapacity);

for (int i = 0; i < 1000; i++) {

    int value = rand.nextInt(maxValue);

    System.out.println("Getting result for value " + value);
    int result = cache.calculate(value);

    System.out.println("Result: " + result);
    System.out.println();
}
```

6. The output should be something like:

```
Getting result for value 8
Expensive calculation: Started for value x=8
Expensive calculation: Done
Result: 16

Getting result for value 7
Expensive calculation: Started for value x=7
Expensive calculation: Done
Result: 14

Getting result for value 7
Using cached data
Result: 14
```

As you notice above for value 7, the cache is reusing the result and no call is made to the calculator, saving us valuable time! Making our customer happy!

7. Start by creating the **SimpleCache** class

8. Add the following **calculate** method to the class as a starting point:

```
public int calculate(int key) throws InterruptedException {

   Calculator calc = new Calculator();
   int result = calc.doComplexCalculation(key);

   return result;
}
```

(The method above just forwards the request to the calculator as a starting point)

9. Your job is to rewrite the method above so that the cache remembers previous calculations.

   The pseudo-code for the necessary logic to write above is:

```
Check if the requested value to calculate is in the cache array
If found, then
    return the stored value
else
    call the calculator and do the calculation
    Add the result to the cache array at a suitable position
    return the result
```

   Questions to ask you self:

   - How to store the data in the cache?
   - How to remember multiple items?
   - What if the cache array is full? Perhaps replace an existing random item?
   - How to add an item to the array?
   - How to lookup if a given item is in the array?

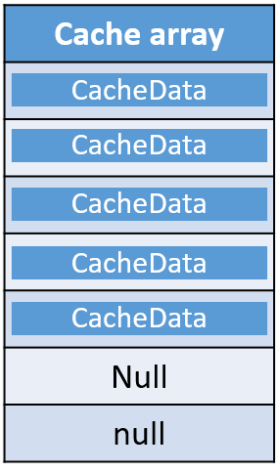
10. To store the data in the cache it can be useful to create a **private array** of **CacheData**. The CacheData class could be implemented as:

```
public class CacheData {
    public int key;
    public int value;
}
```

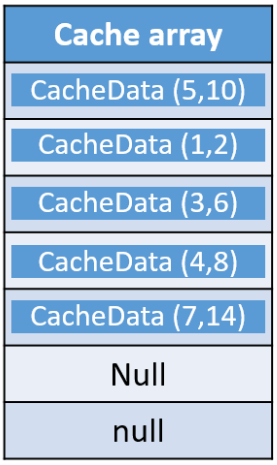
11. Add a **constructor** that take an integer representing the **capacity** of the cache and initialize the array to this value. The capacity will represent the number of unique values the cache will be able to remember the result for. Like

```
cache = new CacheData[capacity];
```

12. This means that the cache array could contain data like:



Where each **CacheData** entry contains the result for a given value, like:



For example, where 5 is the input value and 10 is the calculation result.

Hints:

- Add a private helper method that will look-up if a given value exist in the cache or not.
- Add a private AddValueToCache helper method.

Stretch tasks

If you want more challenges you can try to:

- Add expiration, so that items in the cache will be automatically removed from the cache after a certain time
- Try to implement different cache replacement policies https://en.wikipedia.org/wiki/Cache_replacement_policies
- Study the issue of **time-complexity** and explore the bottlenecks in the current design and possible optimizations https://en.wikipedia.org/wiki/Time_complexity

Questions and concepts to study further on your own:

- Single responsibility principle
- Caching of data
- Does your CPU have its own cache?
- Does your hard-drive have a cache?
- Does your browser have a cache?