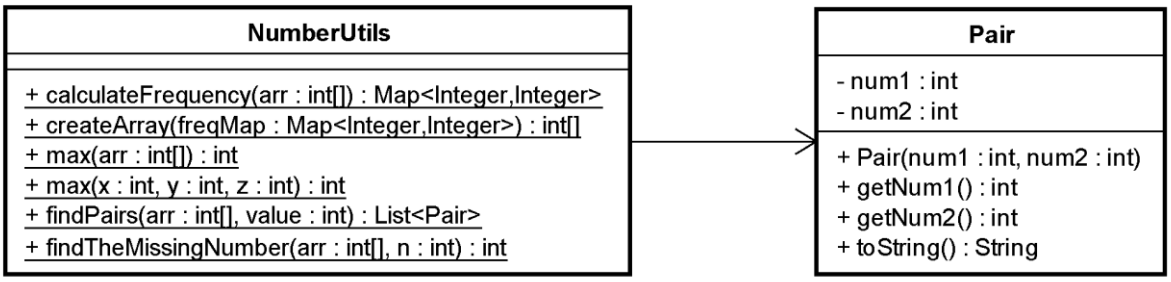
Exercise: Arrays

1. Consider the following class diagram:



Implement the `NumberUtils` class that has the following **static** methods.

a. `calculateFrequency`: returns the number of occurrences of each number in the input parameter, `arr`. For example, if the input parameter is {3,2,2,1,2}, the frequency (number of occurrences) will be:

Number	Frequency
1	1
2	3
3	1

b. `createArray`: returns a new array whereby each integer value will appear multiple times based on the frequency specified in the input parameter `freqMap`.

c. `max`: returns the largest value from the input parameter(s).

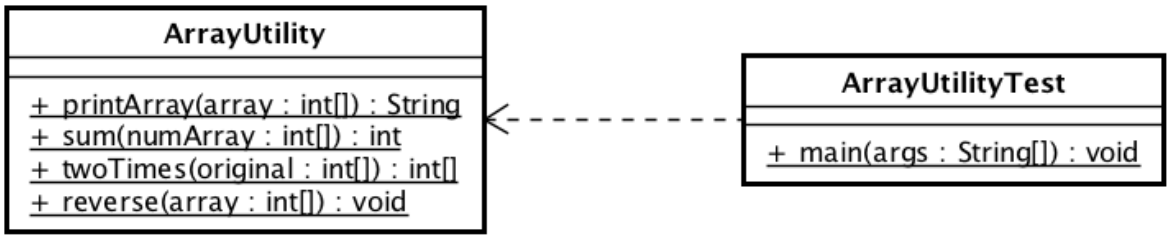
d. `findPairs`: returns all unique pair of integers whose sum is equal to the input parameter value. For example, if input integer array is {2, 6, 8, 3, 1, 1, 2} and the value is 9, output should be [[6, 3], [8, 1], [8,1]]

e. `findTheMissingNumber`: Given 2 input parameters, an integer array that contains numbers from 1 to n (unique values, not sorted) and n (the other input parameter). There is one missing value in the input array (in the range of 1 to n). Find the missing value and return it. For example:

Input	**n**	**Missing number**
{1,2,4,5}	5	3
{2,3,4,5,6}	6	1

Test your code with the given corresponding Test files. Study the test code before implementing the methods.

2. Consider the following class diagram:



Write the `ArrayUtility` class

a. The `printArray` method will return a String in the following format

```
"[<num1>, <num2>, …, <numN>]"
```

b. The `sum` method will return the sum (<num1> + <num2> + … + <numN>) of all the numbers in `numArray`.
c. The `twoTimes` method will return a new array whose elements is twice the value of the `original` array. For example, if the `original` array is `{1,2,3}`, the new array contains the value `{2,4,6}`.
d. The `reverse` method will modify the existing array such that the order of the elements is reversed. For example, if the array is {1,2,3}, then after the `reverse` method is invoked, the array will be {3,2,1}.

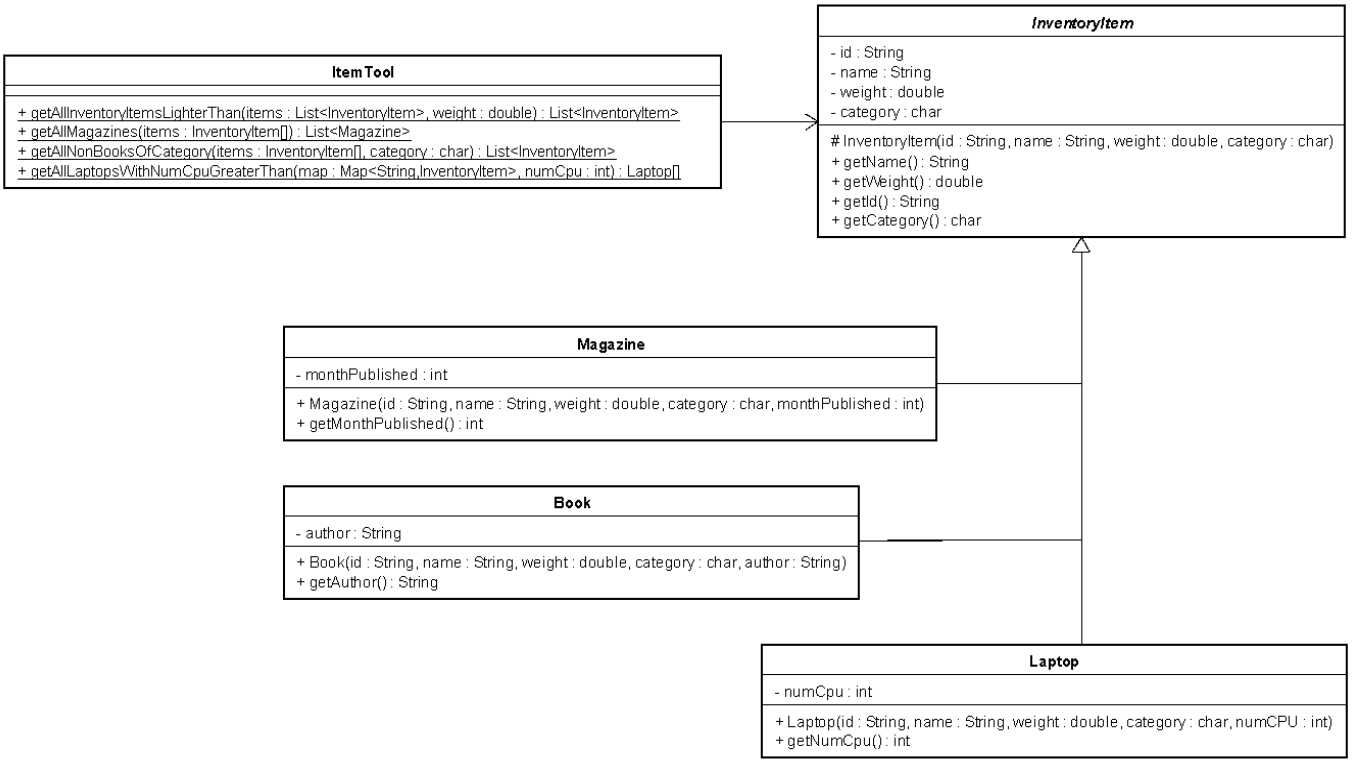
The output of the `ArrayUtilityTest` is as follows:

```
Test sum:
Passed in null: 0
Passed in empty array: 0
Passed in {1,2,4,7,6,8,9}: 37

Test twoTimes:
Passed in null: null
Passed in empty array: []
passing in: [1, 2, 4, 7, 6, 8, 9]
getting out: [2, 4, 8, 14, 12, 16, 18]

Test reverse:
Before reverse: [1, 3, 7, 4, 9]
After reverse: [9, 4, 7, 3, 1]
```

3. Study the class diagram below:



Implement the class `ItemTool`. `ItemTool` has the following **static** methods:

a. `getAllInventoryItemsLighterThan`: returns a new List object containing the `InventoryItem` objects with weight lower than the specified weight.

b. `getAllMagazines`: Returns all the `Magazine` objects contained in `items`.

c. `getAllNonBooksOfCategory`: Returns a new List of `InventoryItem` objects (that are NOT Book objects) with the specified category in `items`.

d. `getAllLaptopsWithNumCpuGreaterThan`:
    i. Returns an array containing only the `Laptop` objects with the number of CPU greater than the specified `numCpu`.

A test class called `ItemApp`.java is provided; you can use it to check if you have written the `ItemTool` class correctly. This is the output when `ItemApp` runs:

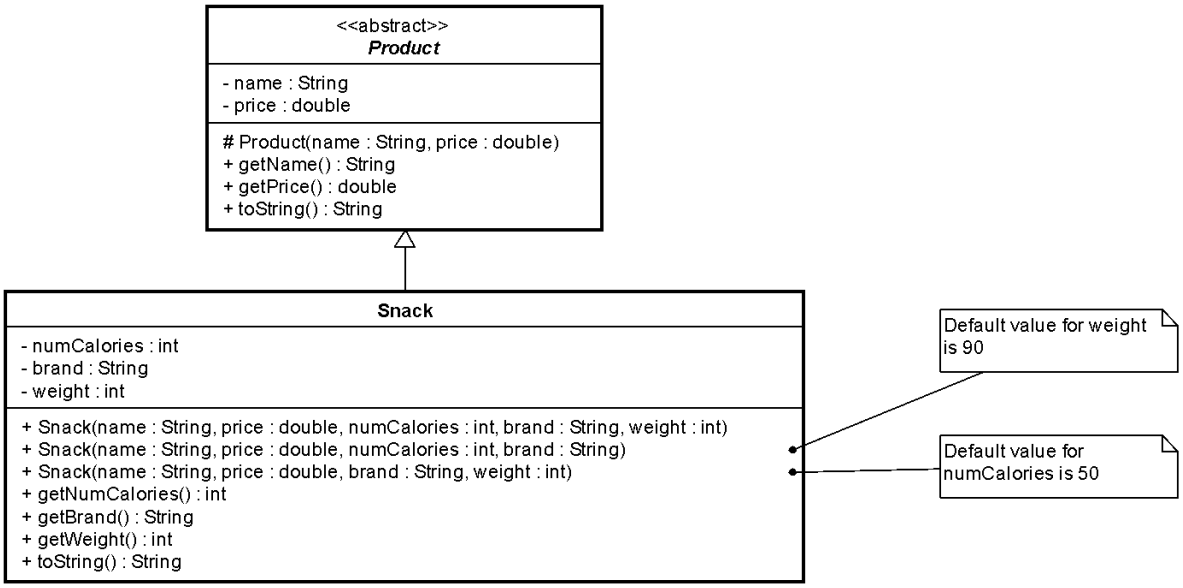
```
Items Lighter than 700:
B001
B002
B003
B004
L005
M001
M002
M003
M004
M005

All Magazines:
M001
M002
M003
M004
M005

All non-book of Category:
L001
M001

Laptops with more than 2 cpu:
L003
L002
```

4. Consider the following class diagram:



Implement the `Product` and `Snack` class. Do make use **of constructor chaining** for the `Snack` class. The output of SnackTest expected is as follows:

```
"Kit kat" price=$5.60 [numCalories=400,brand=Nestle,weight=250]
"Meat Bun" price=$1.20 [numCalories=200,brand=Kong Guan,weight=90]
"Fruits & Nuts Fusion" price=$6.00 [numCalories=50,brand=Tai Sun,weight=150]
```