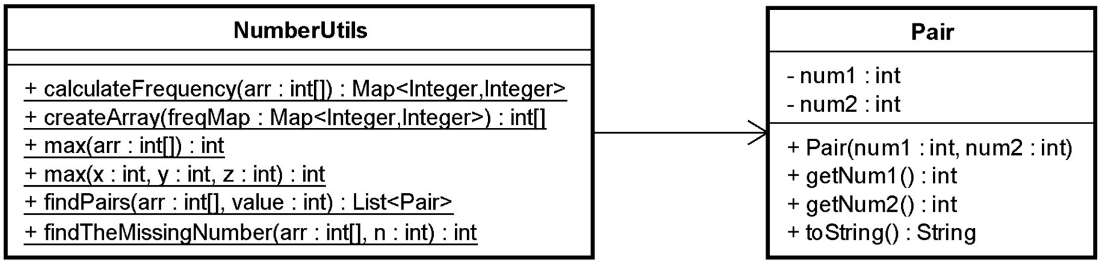
Exercise

Resource: http://blue.smu.edu.sg/array-resource.zip

1. Consider the following class diagram:


NumberUtils
+ calculateFrequency(arr : int[]) : Map<Integer,Integer>
+ createArray(freqMap : Map<Integer,Integer>) : int[]
+ max(arr : int[]) : int
+ max(x : int, y : int, z : int) : int
+ findPairs(arr : int[], value : int) : List<Pair>
+ findTheMissingNumber(arr : int[], n : int) : int
Pair
- num1 : int
- num2 : int
+ Pair(num1 : int, num2 : int)
+ getNum1() : int
+ getNum2() : int
+ toString() : String

Implement the `NumberUtils` class.

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 of specified in the input parameter `freqMap`.

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

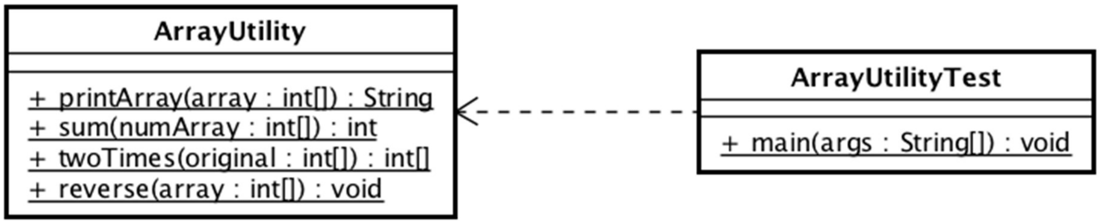
d. `findPairs`: returns all unique pair of integers whose sum is equal to the input parameter value. For example, if input integer array is {2, 6, 8, 3, 1, 1, 2} and the value is 9, output should be [[6, 3], [8, 1], [8,1]]

e. `findTheMissingNumber: Given an` integer array that contains numbers from 1 to n (unique values, not sorted) where n is one of the input parameter. There is one missing value in the 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 test classes.

2. Consider the following class diagram:


ArrayUtility
+ printArray(array : int[]) : String
+ sum(numArray : int[]) : int
+ twoTimes(original : int[]) : int[]
+ reverse(array : int[]) : void
ArrayUtilityTest
+ main(args : String[]) : void

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}.

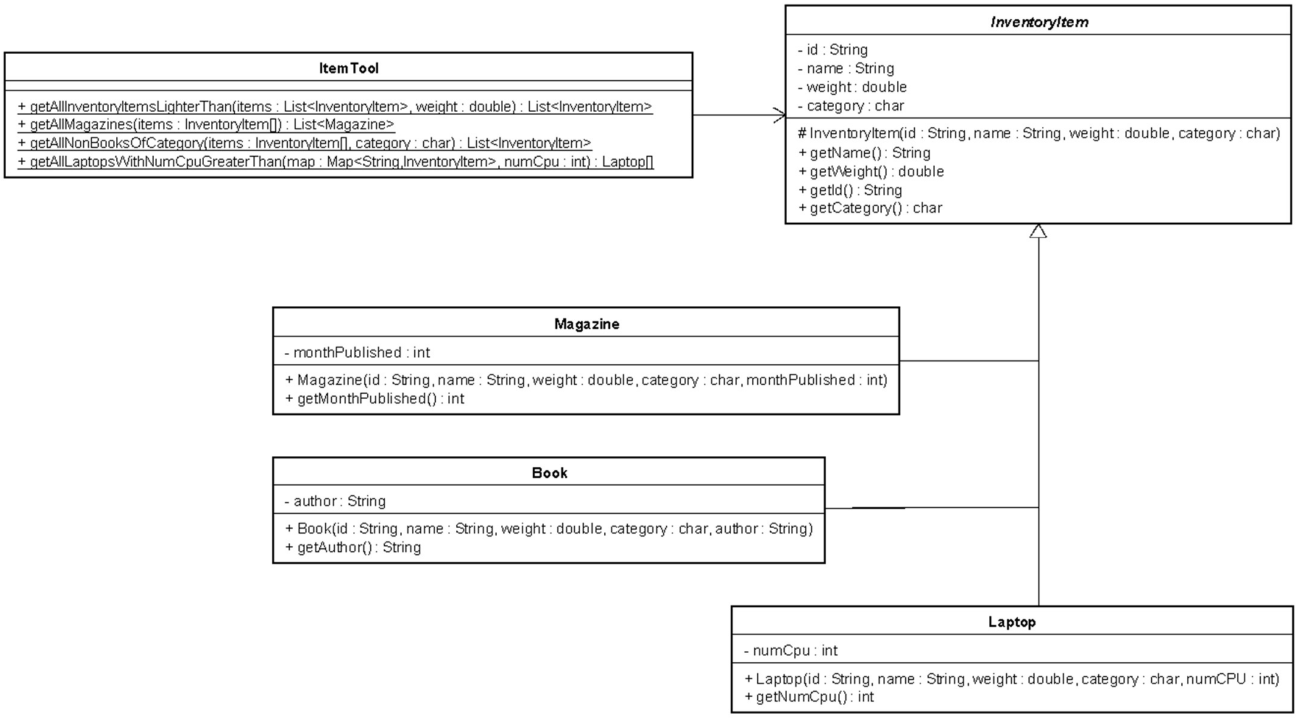
The output of the `ArrayUtilityTest` is as follows:

```
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:


ItemTool
+ getAllInventoryItemsLighterThan(items : List<InventoryItem>, weight : double) : List<InventoryItem>
+ getAllMagazines(items : InventoryItem[]) : List<Magazine>
+ getAllNonBooksOfCategory(items : InventoryItem[], category : char) : List<InventoryItem>
+ getAllLaptopsWithNumCpuGreaterThan(map : Map<String,InventoryItem>, numCpu : int) : Laptop[]
InventoryItem
- id : String
- name : String
- weight : double
- category : char
# InventoryItem(id : String, name : String, weight : double, category : char)
+ getName() : String
+ getWeight() : double
+ getId() : String
+ getCategory() : char
Magazine
- monthPublished : int
+ Magazine(id : String, name : String, weight : double, category : char, monthPublished : int)
+ getMonthPublished() : int
Book
- author : String
+ Book(id : String, name : String, weight : double, category : char, author : String)
+ getAuthor() : String
Laptop
- numCpu : int
+ Laptop(id : String, name : String, weight : double, category : char, numCPU : int)
+ getNumCpu() : int

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 specifiedWeight.

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 specifiedCatogory in items.

d. getAllLaptopsWithNumCpuGreaterThan:

   i. Returns an array containing only the Laptop objects with the number of CPU greater than the specifiedNumCpu.

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:

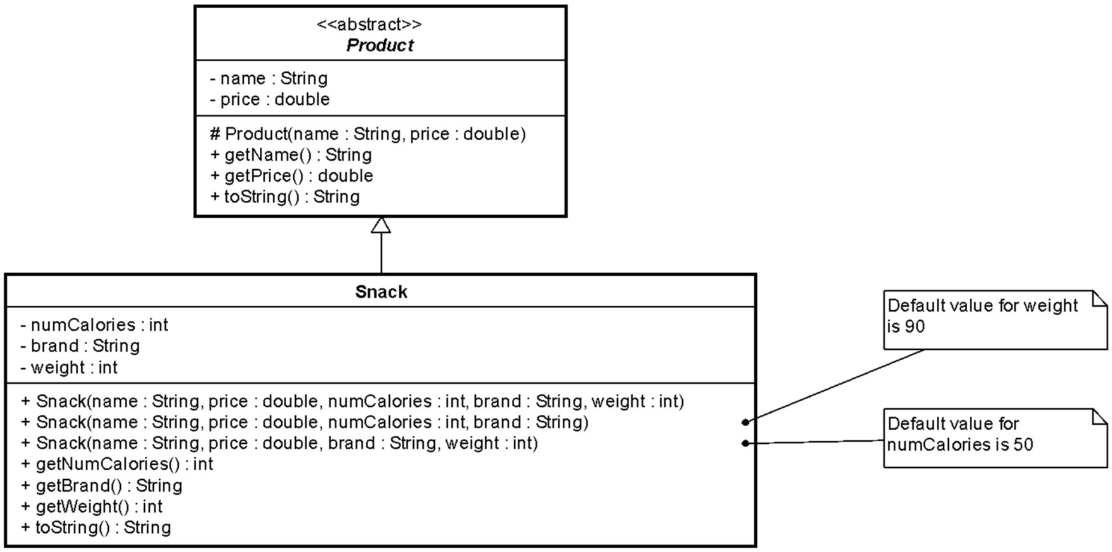
```
Items Lighter than 700:
B001
B002
B003
B004
L005
M001
M002
M003
M004
M005

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:



<<abstract>>
Product
- name : String
- price : double
# Product(name : String, price : double)
+ getName() : String
+ getPrice() : double
+ toString() : String
Snack
- numCalories : int
- brand : String
- weight : int
+ Snack(name : String, price : double, numCalories : int, brand : String, weight : int)
+ Snack(name : String, price : double, numCalories : int, brand : String)
+ Snack(name : String, price : double, brand : String, weight : int)
+ getNumCalories() : int
+ getBrand() : String
+ getWeight() : int
+ toString() : String
Default value for weight is 90
Default value for numCalories is 50

Implement the `Product` and `Snack` class. Do make use of constructor chaining for the `Snack` class. The output of SnackTest 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]
```

5. Implement the class `MyArrayList` that behaves in similar way as `ArrayList` class. The class should contains the following methods:
   a. `add(E e):` Appends the specified element to the end of this list.
   b. `get(int index):` Returns the element at the specified position in this list
   c. `remove(int index):` Removes the element at the specified position in the list. Shifts any subsequent elements to the left (subtract one from their indices).
   d. `size():` Returns the number of elements in this list.

```
public class MyArrayList <E> {
    private int count;
    private Object[] array;

    // …
}
```

a. Initial: The class will start off by creating an initial size of 3.

count = 0

b. After adding element "apple"

count = 1

"apple"		

c. After adding element "orange"

count = 2

"apple"	"orange"	

d. After adding element "pear"

count = 3

"apple"	"orange"	"pear"

e. After adding element "banana"

There is no more empty slot in the array to add the new element. Therefore, a new array is created. The new size of the array is calculated using this formula: (old capacity x 3)/2 + 1. The elements from the old array is then copied into the new array, before adding the new element ("banana") behind.

count = 4

"apple"	"orange"	"pear"	"banana"	

6. Internally, a HashMap is implemented via the use of a hash table using the data structure of arrays. You can read more about hash tables here: https://www.hackerearth.com/practice/data-structures/hash-tables/basics-of-hash-tables/tutorial/. Implement the class MyHashMap using separate chaining (refer to link earlier).

   Define the following methods:

   a. get(key):
   b. put(key,value):
   c. remove(key):
   d. containsKey(key):
   e. size().

   Do not use *any* of Java's generic data structures in the `java.util` package. Assume that both keys and values are of type Object. Every Object has a hash code, so at least you don't have to define your own hash functions.