**Object Oriented Development**

Module 5: Methods

**This document includes the answers to the exercises**

© FDM Group Ltd 2020. All Rights Reserved.

Any unauthorised reproduction or distribution in part  
or in whole will constitute an infringement of copyright.

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Author | Comments |
| 1.0 | 15 / 10 / 20 | Nick Lawton | First draft |
| 2.0 | 22/06/21 | Nick Lawton | Shortened to fit into half a day |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## **Please note:**

Be careful about looking at the solutions too quickly; make sure you’ve given yourself time to wrestle with the concepts you just learned before looking at a solution. Also, there are several ways to solve many of the exercises, and the solutions only show one possible way to complete each exercise.

# Section 1 – Modularising previous exercises

In this section’s exercises, you should ensure that your method uses the ‘static’ keyword in its header. You should call the method from the main method in the same class.

## Alien colours

You’re going to modularise your code from exercise 1.3 in the If statements module.

Create a method called getPoints. The method should take the colour of the alien as an argument and should return the number of points you get for that colour as an int.

Copy your code from the original exercise into your method and modify it so that it returns only the number as an **int** and not the full sentence. If an invalid colour is entered the method should return 0.

The code in your main method should look like this:

int points = getPoints(“green”);

System.out.println(“You’ve scored “+points+” points!”);

## Summing numbers

Create a method called sumOfNumbersUpTo. It should take an int as an argument and return a long.

Take your code from the Loops module exercise 2.2. Put it into the method and modify it so that it counts up to whatever number you entered as an argument.

For instance:

* If you pass in 5 as an argument the method should return 15 (1+2+3+4+5)
* if you pass in 10 as an arugment the method should return 55.

Display the result in the main method.

## 1.3 Times table

Create a void method called displayTimesTable. It should take an int as an argument and display the times table for that int in the same format as in Loops exercise 2.4.

For instance if the argument is 3 the method should display the 3 times table. If the argument is 5 the method should display the 5 times table.

The method shouldn’t return anything.

# Section 2 - Methods with test cases

In this section’s exercises you’ll be using pre-written test cases to check if your code works.

Clone the project from Git and then import into Eclipse:

<https://git.fdmgroup.com/java-trainers-shared/ood-week-1-exericse-5-testcases.git>

The method header for each of the 8 methods in this section has already been created. You need to complete the code in the method and then run the test.

## 2.1 Leap years

Write a method called leapYear. It should take an int argument containing a year and return a boolean which is true if the year is a leap year.

Leap years are divisible by 4. However the first year of a century is not a leap year unless the year is divisible by 400.

E.g. 2016 is a leap year so should return true

2019 is not a leap year so should return false

2000 is a leap year so should return true

1900 is not a leap year so should return false

## 2.2 Array contains

Write a method called arrayContains. It should take two arguments: a String array and a String. The method should return a Boolean: true if the array contains the String and false otherwise.

## 2.3 Array frequency

Write a method called arrayFrequency. It should take two arguments: an int array and a int. The method should return an int containing the number of times the int occurs in the array.

## 2.4 Highest number

Write a method called maxNumber. It should take an int array as an argument. It should return an int containing the highest number in the array. Note that numbers in the array could be positive or negative. So in this array: [-7,-4,-9] the highest number would be -4.

## 2.5 Extracting even numbers

Write a method called extractEvenArray. It should take an int array as an argument. It should return an int array which contains all of the even numbers from the original array. The returned array should not be any bigger than the size required to hold the even numbers. The even numbers in the returned array should be in the same order as in the original array. For example:

With this array as an argument: [2,5,3,7,6,7,3,4,3]

The method would return: [2,6,4]

## Reversing an array

Write a method called reverseStringArray. It should take a String[] array as an argument and should return a reversed copy of the array. The original array should not be reversed.

## Finding anagrams

Write a method called isAnagram. It should take two Strings as arguments. If the Strings are anagrams of each other the method should return the boolean true, otherwise it should return false.

## Median number

Write a method called medianNumber which takes an array of doubles and returns the median value.

Argument:

* An array of doubles

Return value:

* A double containing the median value from the array

For example:

* Array [1.0, 2.0, 3.0, 4.0, 5.0] would return 3.0 as this is the middle value
* Array [1.0, 2.0, 3.0, 4.0] would return 2.5 as this is the mean of the two middle values (2.0 and 3.0)
* Array [7.4, 3.9, 1.2, 9.7, 2.3] would return 3.9 as this is the middle value
* Array [7.4, 3.9, 1.2, 9.7, 2.3, 8.2] would return 5.65 as this is the mean of the two middle values (3.9 and 7.4)

ANSWERS

## Alien colours

You’re going to modularise your code from exercise 1.3 in the If statements module.

Create a method called getPoints. The method should take the colour of the alien as an argument and should return the number of points you get for that colour as an int.

Copy your code from the original exercise into your method and modify it so that it returns only the number as an **int** and not the full sentence.

The code in your main method should look like this:

int points = getPoints(“green”);

System.out.println(“Your’ve scored “+points+”!”);

**public** **static** **int** getPoints(String alienColour){

**if** (alienColour.equals("green")){

**return** 5;

}

**if** (alienColour.equals("red")){

**return** 10;

}

**if** (alienColour.equals("yellow")){

**return** 15;

}

**return** 0;

}

## Summing numbers

Create a method called sumOfNumbersUpTo. It should take an int as an argument and return a long.

Take your code from the Loops module exercise 2.2. Put it into the method and modify it so that it counts up to whatever number you entered as an argument.

For instance:

* If you pass in 5 as an argument the method should return 15.
* if you pass in 10 as an arugment the method should return 55.

Display the result in the main method.

**public** **static** **long** sumOfNumbersUpTo(**int** endNumber){

**long** total = 0;

**for** (**int** count = 1; count <= endNumber; count++){

total += count;

}

**return** total;

}

## 1.3 Times table

Create a method called displayTimesTable. It should take an int as an argument and display the times table for that int in the same format as in Loops exercise 2.4.

For instance if the argument is 3 the method should display the 3 times table. If the argument is 5 the method should display the 5 times table.

The method shouldn’t return anything.

**public** **static** **void** displayTimesTable(**int** table){

**for** (**int** count = 1; count<= 12; count++){

System.***out***.println(count+" x "+table+" = "+(count\*table));

}

}

# Section 2 - Methods with test cases

## 2.1 Leap years

Write a method called leapYear. It should take an int argument containing a year and return a boolean which is true if the year is a leap year.

Leap years are divisible by 4. However the first year of a century is not a leap year unless the year is divisible by 400.

E.g. 2016 is a leap year so should return true

2019 is not a leap year so should return false

2000 is a leap year so should return true

1900 is not a leap year so should return false

**public** **static** **boolean** leapYear(**int** year) {

**if** (year%4 != 0) {

**return** **false**;

}

**if** (year%400 == 0) {

**return** **true**;

}

**if** (year%100 == 0) {

**return** **false**;

}

**return** **true**;

}

## 2.2 Array contains

Write a method called arrayContains. It should take two arguments: a String array and a String. The method should return a Boolean: true if the array contains the String and false otherwise.

**public** **static** **boolean** arrayContains(String[] array, String string) {

**for** (String element : array) {

**if** (element.equals(string)) {

**return** **true**;

}

}

**return** **false**;

}

## 2.3 Array frequency

Write a method called arrayFrequency. It should take two arguments: an int array and a int. The method should return an int containing the number of times the int occurs in the array.

**public** **static** **int** arrayFrequency(**int**[] array, **int** number ) {

**int** count = 0;

**for** (**int** element : array) {

**if** (element == number) {

count ++;

}

}

**return** count;

}

## 2.4 Highest number

Write a method called maxNumber. It should take an int array as an argument. It should return an int containing the highest number in the array. Note that numbers in the array could be positive or negative. So in this array: [-7,-4,-9] the highest number would be -4.

**public** **int** maxNumber(**int**[] array) {

Arrays.*sort*(array);

**int** highestIndex = array.length-1;

**int** maximum = array[highestIndex];

**return** maximum;

}

## 2.5 Extracting even numbers

Write a method called extractEvenArray. It should take an int array as an argument. It should return an int array which contains all of the even numbers from the original array. The returned array should not be any bigger than the size required to hold the even numbers.

For example, with this array as an argument: [2,5,3,7,6,7,3,4,3]

The method would return: [2,6,4]

**public** **static** **int**[] extractEvenArray(**int**[] array) {

**int** newArrayLength = 0;

**for** (**int** number : array) { // determine length of new array

**if** (number%2 == 0) {

newArrayLength ++;

}

}

**int**[] evenNumbers = **new** **int**[newArrayLength]; // create new array

**int** evenNumberIndex = 0;

**for** (**int** number : array) { // add numbers to new array

**if** (number%2 == 0) {

evenNumbers[evenNumberIndex] = number;

evenNumberIndex++;

}

}

**return** evenNumbers;

}

Using Java 8 syntax this can be done much more simply:

**public** **static** **int**[] extractEvenArray(**int**[] array) {

ArrayList<Integer> evens = **new** ArrayList<Integer>();

**for** (**int** number : array) {

**if** (number%2 == 0) {

evens.add(number);

}

}

// convert ArrayList<Integer> to int[] then return int[]

**return** evens.stream().mapToInt(i->i).toArray();

}

Or even in a single line:

**public** **static** **int**[] extractEvenArray(**int**[] array) {

**return** Arrays.*stream*(array).filter(number -> number%2 == 0).toArray();

}

## Reversing an array

Write a method called reverseStringArray. It should take a String[] array as an argument and should return a reversed copy of the array. The original array should not be reversed.

**public** **static** String[] reverseStringArray(String[] array) {

**int** length = array.length;

String[] reversed = **new** String[length];

**for** (**int** index = 0; index < length; index++) {

**int** reversedIndex = length - 1 - index;

reversed[reversedIndex] = array[index];

}

**return** reversed;

}

## Finding anagrams

Write a method called isAnagram. It should take two Strings as arguments. If the Strings are anagrams of each other the method should return the boolean true, otherwise it should return false.

**public** **static** **boolean** isAnagram(String string1, String string2) {

**char**[] string1Array = string1.toCharArray();

**char**[] string2Array = string2.toCharArray();

Arrays.*sort*(string1Array);

Arrays.*sort*(string2Array);

string1 = **new** String(string1Array);

string2 = **new** String(string2Array);

**return** string1.equals(string2);

}

## Median number

Write a method called medianNumber which takes an array of doubles and returns the median value.

Argument:

* An array of doubles

Return value:

* A double containing the median value from the array

For example:

* Array [1.0, 2.0, 3.0, 4.0, 5.0] would return 3.0 as this is the middle value
* Array [1.0, 2.0, 3.0, 4.0] would return 2.5 as this is the mean of the two middle values (2.0 and 3.0)
* Array [7.4, 3.9, 1.2, 9.7, 2.3] would return 3.9 as this is the middle value
* Array [7.4, 3.9, 1.2, 9.7, 2.3, 8.2] would return 5.65 as this is the mean of the two middle values (3.9 and 7.4)

**public** **double** medianNumber(**double**[] numbers) {

Arrays.*sort*(numbers);

**int** middleIndex = numbers.length / 2;

**if** (numbers.length%2 == 0) {

**return** (numbers[middleIndex]+numbers[middleIndex-1]) / 2;

} **else** {

**return** numbers[middleIndex];

}

}

Click on the link to give feedback on this exercise:

<https://forms.office.com/Pages/ResponsePage.aspx?id=glOkWCW86EGcIlkUGYi-mhn5SZBIBvtIk9fURIMGJDZUQ1M4NUVEQkU1MTdCMlhQWkJCMVdIUlFSRy4u>