**Object Oriented Development**

Module 5: Methods - advanced

© 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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Section 1

Download the project from Git and import into Eclipse:

<https://git.fdmgroup.com/java-trainers-shared/ood-week-1-module-5-advanced-test-cases.git>

The method headers for the questions are supplied. You need to complete the code in the methods and run the testcases.

## Prime numbers

A prime number can’t be divided by anything but itself or 1. The first 5 prime numbers are 2, 3, 5, 7, 11.

To find out if a number is a prime you could try dividing it by every number below it. But this would be very inefficient.

A more efficient way to do this would be to see if it could be divided by any of the prime numbers before it. For instance 11 can’t be divided by 2, 3, 5 or 7. Therefore it’s a prime number.

Write a method called primeNumbers which takes an int as an argument. The method should return an ArrayList containing all of the prime numbers up to and including the argument.

e.g. if the argument is 12 the method should return a list containing: 2, 3, 5, 7, 11

If the argument is 50 the method should return a list containing: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

## Bubble sort

You’re going to write a method called bubbleSort which performs a bubble sort on an array.

A bubble sort is a simple way of sorting an array. It works by looking at pairs of variables and flipping them if they’re in the wrong order.

For instance let’s say we want to sort the following numbers: 3,2,1,5,4

We start with the first pair: 3 and 2. They’re the wrong way round so we flip them giving:

2,3,1,5,4

We now look at the next pair: 3 and 1. They’re the wrong way round so we flip them giving:

2,1,3,5,4

The next pair 3 and 5 are the right way round so we leave them.

The final pair 5 and 4 are the wrong way round so we flip them giving:

2,1,3,4,5

We now repeat the process starting with the first pair 2 and 1. They’re the wrong way round so we flip them giving:

1,2,3,4,5

We continue checking pairs of numbers until we can go through the whole array without needing to flip any.

Write a method called sortIntArray. It should take an int array as an argument and return a sorted copy of the array.

## Simple encoder

Write a method called encode, which takes a String as an argument and returns an encoded version of the String.

The encoding algorithm will be very simple:

* Each letter in an odd position will be moved forward one place in the alphabet. So ‘a’ would become ‘b’ and ‘y’ would become ‘z’. The last letter of the alphabet ‘z’ would loop round to become ‘a’.
* Each letter in an even position will be moved backward one place in the alphabet. So ‘b’ would become ‘a’ and ‘z’ would become ‘y’. The first letter of the alphabet ‘a’ would loop round to become ‘z’.
* Spaces will not be changed

Assume that the String only contains lower case letters and spaces.

Assume that the first character in the String is character 1 (odd).

e.g. Argument “abcd” would return encoded String “badc”

Argument “two words” would return “uvp xnsct”

## Harder encoder

You’re going to write another method called encode, which returns an encrypted String. This time the encrytption is going to be stronger.

Once again odd characters will be moved forward and even characters will be moved back. The difference this time is that the number of characters they will be moved forward or back will be determined by their position in the original String.

The alphabet will be treated as circular so that ‘a’ minus two characters will be ‘y’. ‘z’ plus three characters will be ‘c’.

For instance argument “abcd” will return “bzfz”

e.g. a + 1 character = b

b – 2 characters = z

c + 3 characters = f

d – 4 characters = z

All letters should be converted to lower case so: “ABCD” will return “bzfz”

The character in the 26th position will be cycled through the entire alphabet and so will remain the same. i.e. if the character in position 26 of the original String is ‘z’ then the character in position 26 of the encoded String will also be ‘z’.

For example:

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa (52 x ‘a’)

will return

bydwfuhsjqlonmpkritgvexczabydwfuhsjqlonmpkritgvexcza

Notice that the last 26 letters are exactly the same as the first 26 letters.

Spaces will not be encoded.

## Rotating arrays

You’re going to write a method called rotateArray which rotates values in an array of ints. The method will return a rotated copy of the original array. The method will take 2 arguments:

* The array which is going to be rotated.
* An int containing the number of positions the characters will be rotated.

The method will rotate the values to the right by the number specified in the argument.

So if the array is [1,2,3,4,5]

* Rotating it one position would give [5,1,2,3,4]
* Rotating it two positions would give [4,5,1,2,3]
* Rotating it three positions would give [3,4,5,1,2]
* Rotating it four positions would give [2,3,4,5,1]
* Rotating it five positions would give [1,2,3,4,5]
* Rotating it six positions would give [5,1,2,3,4]

The array can be any length.

The number of rotations can be any positive number which fits into an int.

## Moon rover

You’re going to write a method called roverMove which simulates the movement of a moon rover vehicle within a grid.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 |
| 12 | 13 | 14 | 15 |

The starting position is zero. You can move the rover by using the commands: “UP”, “DOWN”, “LEFT”, “RIGHT”

The rover can’t move outside of the grid. For example on a 4x4 grid if the rover is currently in position 3:

* LEFT takes it to position 2
* RIGHT leaves it in position 3
* DOWN takes it to position 7
* UP takes leaves it in position 3

If the rover is currently in position 12:

* LEFT leaves it in position 12
* RIGHT takes it to position 13
* DOWN leaves it in position 12
* UP takes it to position 8

Arguments:

* An int containing the size of the grid
* An array of Strings containing the commands

Return value:

* An int containing the final grid position after the commands have been followed.

For example:

On grid size 4, the following sequence of commands:

* LEFT, DOWN, RIGHT, DOWN, LEFT, DOWN, UP

Would leave the rover in position 8

On grid size 5 the following sequence of commands:

* "RIGHT",”UP, "DOWN", "RIGHT", "RIGHT","RIGHT",“ RIGHT",”RIGHT”

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 |

Would leave the rover in position 9.

## Valid UK postcode

You’re going to write a method called validPostcode. The method will take a String argument containing the postcode to be verified. It will return a boolean: true if the postcode is valid and false otherwise.

A valid UK postcode will contain between 5 and 8 characters. It will be made up of two parts which may or may not be separated by a space.

A12BC is a valid postcode with 5 characters and no space

AB12 3CD is a valid postcode with 8 characters including a space.

The first part of a postcode can be between 2 and 4 characters long and will have one or two letters followed by one or two numbers:

e.g. AB12 or AB1 or A12 or A1

The second part of a postcode will always be formed of 1 number followed by 2 letters.

e.g. 1AB

A valid postcode could contain upper case or lower case letters.

This method could be written using syntax we’ve covered in this course. However it would be quite long and inefficient.

A more efficient way to write this is to use the matches() method from the String class with ‘regex’ . We haven’t covered regex in the course so you’ll need to research how to use it.

## Valid password

You’re going to write a method called validPassword. The method will take a String as an argument which contains a password. It will return a boolean: true if the password is valid, false otherwise.

A valid password should have at least 7 characters. It should contain 3 of the following 4 characters: upper case letter, lower case letter, number, symbol.

The most efficient way to do this would be using regex.

Click on the link to give feedback on this exercise:

<https://forms.office.com/Pages/ResponsePage.aspx?id=glOkWCW86EGcIlkUGYi-mhn5SZBIBvtIk9fURIMGJDZUM1NLWDBaWTVSSVRCTExCU1dZMzQwTlBENS4u>