BootCamp 2016
Object-Oriented Programming in Java

Here you may find exercise composed from all the taught courses during these two weeks (10 - 22 of Oct. 2016). The difficulty of each exercise is written at the end of the exercise description. After the exercise titles you may find in parenthesis what type of exercise it is. We have added many exercises for practise, thus, if you may not be able to solve them during the Bootcamp you may try solving them at home.
Some exercises are providing source code that you may use in order to accomplish the task, also, test case can be found inside those source codes (if your results are correct you may proceed). In order to use the existing source code you can create a new project and copy paste the source code files (.java) inside your IDE's, i.e., eclipse, IntelliJ, project manager.

Exercise 1 : Poll Analysis

(Arrays)

We asked from 40 students to rate the quality of the school canteen's meals. The rating was from 1 to 10, where 1 means awful and 10 means delicious. Put the 40 answers in an (int) table and summarize the results. Create a class called Poll and a main method where to write your program.

Answers to put into the table:

1, 2, 6, 4, 8, 5, 9, 7, 8, 10, 1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6, 5, 6, 7, 5, 6, 8, 4, 6, 8, 10.

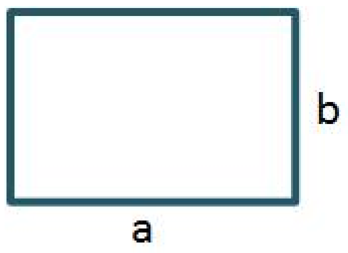
The output of your program should be like:

```
Rating Frequency
1      2
2      2
3      2
4      2
5      5
…      …
```

(Exercise difficulty: Normal)

Exercise 2 : The Rectangle shape

(Creating Classes)



Create a class that represents the Rectangle geometrical shape. A rectangle is defined by the length of its sides a and b. Implement the following basic methods:

1. Getters and Setters
2. The default constructor and an overloaded constructor
3. toString()

Also provide your class with the following functionalities:

- areEqual - compares two rectangles
- isSquare - checks if a rectangle is a square
- draw - draws the contour of your shape on the console
- copy - creates and returns an exact copy of a rectangle

Create a TestRectangle class with a main method. Create various rectangle objects and test their functionalities!

(Exercise difficulty: Normal)

Exercise 3 : Shapes

(Creating Classes, Inheritance, Polymorphism, Abstraction)

Step 1 : Extend the "Rectangle" exercise by adding an abstract Shape class. The Shape class should have one attribute, the *number of sides* and provide two abstract methods:

- *area* - calculates and returns the area of a Shape, and the
- *perimeter* - calculates and returns the perimeter of a shape.
- *draw* - draws the contour of a shape

Choose an appropriate "access modifier" for the number_of_sides field so that is visible and directly accessible from Shape's subclasses. Do not forget to implement the *getter/setter* methods, the *Constructors* and the *toString* method.

Step 2 : Modify your Rectangle class so that inherits from the Shape class. Modify the code accordingly so that a Rectangle object initializes also the *number_of_sides* field from the superclass. Also implement the two abstract methods inherited from the Shape class.

Step 3 : Create a TestShape class with a *main* method, create Rectangle objects and test their functionalities. Create 2 Rectangle objects using:

```
Shape rer1 = new Rectangle(2,3);
Rectangle rec2 = new Rectangle(3,5);
```

What do you notice for the rec2 methods *areEquals* & *isSquare*? Try to explain this observation.

Step 4 (optional): If you want to practice further in the Inheritance and Polymorphism topics, create a class Circle (like you did in earlier exercises) and implement all the methods that inherits from the class Shape.Then, test its functionalities.

Step 4b : Create an Array that stores Shapes (`Shapes[]`) and fill it with different Rectangles and Circles. Use the *toString* method to show the details of each object in the array.

(Exercise difficulty : Normal)

Exercise 4 : Safe Division

(Creating Classes, Exceptions)

Create a class called Quotient that has a method declared as calcQuotient(int numerator, int denominator). This method returns the quotient of the given numbers and it can throw an ArithmeticException (in case of a divizion with zero). Then, in Quotient create a main method the uses a Scanner object to get keyboard input from the user. In the main method we ask from the user to give two numbers and we use the calcQuotient method to calculate the quotient. If user imports 0 for a denominator, there should be an ArithmeticException. If user imports a string (e.g. “hello”) as denominator, there should be an InputMismatchException. When these exceptions occur we ask from the user to enter again the inputs. (Note that you do not need to create the exceptions, they already exist in Java.)

(Exercise difficulty: Normal)

Exercise 5 : Time comparison

(Creating Classes, Interfaces)

Create a class called TimeComparator that implements Comparator<TimeCom> and compares two objects of type TimeCom (represents time in hours, minutes, and seconds); you need to also create the TimeCom class. In class TimeComparator you create a method called compare(TimeCom time1, TimeCom time2) which will return the difference of two times. If the value of the difference is positive, the first time is greater than the second time, otherwise if the value is negative, second time is greater than the first time. If the value is zero you need to also check the minutes and probably the seconds.

The TimeCom class has three private fields: int hour, int minute, int second. Also, TimeCom has a default constructor and one constructor to initiate the class fields (i.e. constructor’s declaration is TimeCom(int hour, int minute, int second)). The class has getters and setters to access the private fields and a method (called toString) that converts the time into a string (i.e. HH:MM:SS). Keep in mind that when setting a new time, the setters (there is a different set method for accessing each field) checks if hour is 0-23, minute is 0-59, and second is 0-59; if a field is wrong, set the field equal to 0. Finally, there is a method called setTime that sets the time i.e. setTime(int h, int m, int s).

(Exercise Difficulty: Medium)

Exercise 6 : Anagrams

(String manipulation, Arrays)

Check if two given two strings , stringA and stringB, are anagrams or not. For example, anagram string is considered a word which can be written like DOG, DGO, OGD, ODG, GOD, and GDO.
Given Strings should not exceed length of 50 characters. Your application must print “Anagrams” or “Not Anagrams”. Also, comparison must NOT be case sensitive.
NOTE: input strings are given in the main for you and must not be modified but check if your output is equals to expected output.
Download source code from here.

Example 1
Input String A: anagrams
Input String B: msanagrm
Expected output: Not Anagrams

Example 2
Input String A: anagrams
Input String B: msanagra
Expected output: Anagrams

(Exercise difficulty: Normal)

Exercise 7 : BigIntegers

(Non-Primitive types)

For this problem you are called to use BigInteger to sum and multiply huge numbers that cannot be added in any other data type like: int, long int, etc.
You may use this link for understanding BigIntegers.
Download source code from here.

(Exercise difficulty: Normal)

Exercise 8 : Existing Anagrams

(ArrayList, String comparison, File Handling)

Modify the source code that you create in the previous exercise, so that you can find all valid anagrams from a given words.
In order to verify if a generated anagram is valid or not you will need an English dictionary, which you can find in this link. Place the dictionary txt file in your src folder with your java code. Read the dictionary and store it in a data structure (of your own preference). For a given "3-letter word", check if each generated anagram exists in the dictionary and print it. (try the words "dog" and "tea" that can be found in the dictionary)

You can find more information on how to open a file and read its contents in the Oracle's java tutorial.

(Exercise difficulty: Challenging)

If you want to increase the challenge of this exercise, solve it for words without a limitation in the number of the characters. The use of **recursion** is strongly recommended for this solution. You also have to present each valid anagram only once. Check how the Sets work (Specifically, the HashSet java docs)

(Exercise difficulty: Hard)

Exercise 9 : PrimeFactors

(Non-Primitive types)

The prime factors of 13195 are 5, 7, 13 and 29. What is the largest prime factor of the number 600851475143 ?
Note: Consider the fact of using BigIntegers (Result must be: **6857**)

Exercise Difficulty: Medium)

Exercise 10 : The Frogs Contest

(Objects management)

Consider two frogs (objects) starting from x-axis and jumping towards the positive infinite direction. The first frog's starting point is x1 and moves with rate of v1, meters jump, while the

second starts at x2 and moves at v2 rate, meters per jump. Given their starting position and movement rate determine if they will ever land at the same location at the same time.
A single line of input must indicate the x1, v1, x2, and v2. Print YES or NO, respectively.
You may use available source code from here.

Example: 0 3 4 2
Output: YES

Explanation
Frog 1: 0 → 3 → 6 → 9 → 12
Frog 2: 4 → 6 → 8 → 10 → 12

(Exercise Difficulty: Medium)

Exercise 11 : Reading the content of a network packet

(String manipulation, Casting, File Handling)

Scenario:
You are a network security engineer and you have been asked to monitor the messages, exchanged in a company's private network.
You have managed to collect some packages exchanged between users in a binary form (0s and 1s). Your next task is to transform these binary messages to a form readable by humans.
The following figure presents the fields that comprise a network packet (a very simplified version)

Network packet					
Type (8 bits)	Source ip (32 bits)	Destination ip (32 bits)	Size (8 bits)	Data (variable size)	Other info (variable size)

A network package is a sequence of zeros “0s” and ones “1s” that are called **bits**. A group of 8 bits form a **byte**. Each byte represents a number that you can map to the ascii table. The following list briefly describes how a network packet is formed.

- Type - defines the type of the package. Only packages of type "4" (character ‘4’ in the ascii table) are of our interest.
- Source ip - The ip address xxx.xxx.xxx.xxx of the packet's creator.
- Destination ip - The ip address xxx.xxx.xxx.xxx of the recipient.
- Data size - The number of the characters (1 character is represented by 8 bits (1 byte)) of the message that follows in the next field.
- Data - The actual message.
- Other information - Various other information, out of our interest.

Tasks

1. Download the "collected_data.txt" file from this link and save the file preferably in the same folder as your java project.

2. Create a program that reads the file, parses the stored information, and prints the text messages (the Data field of the package) in a form readable by humans (not 0s and 1s)!

Notes:

- Read the Integer class from the Java docs in order to find a way how to transform a sequence (8 bits) of 0s and 1s into the corresponding ascii number.
- Check also the Oracle's java tutorials for the Buffered I/O Methods for Text Files, that will help you open and read the contents of a .txt file.

(Exercise difficulty: Challenging)

Exercise 12 : Find the largest product from 20x20 grid array

(Arrays)

Given a 20x20 grid array find the largest product by multiplying 4 number. The selected number can go up, down, left, right, or diagonal. A source code file is give which reads a file GridMap.txt and create an int array of 20x20 with all the number. You must not read the file, just use the source code which will do everything for you. Source code is here. Grid map is below showing the different ways you can make a product of 4 numbers.

```
08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48
```

Expected output : **70600674**

(Exercise difficulty: Hard)

Exercise 13 : Recursion

(Strings, Recursion)

Recursively find the number of times a particular digit appears in a number. E.g if number is 13563 and the digit is 3, the method should return 2

(Exercise difficulty: Medium to Hard)

Exercise 14 :Bin Packing Problem

(Logic, Arrays, Lists)

Έχουμε ένα σύνολο αντικειμένων με ένα συγκεκριμένο χαρακτηριστικό (ύψος, βάρος κτλ). Κάθε ένα από αυτά τα αντικείμενα θέλουμε να τοποθετηθεί σε κάδους ομοίων χαρακτηριστικών. Σε κάθε επανάληψη του αλγορίθμου αποθηκεύεται ένα αντικείμενο που θα πρέπει να έχει διαστάσεις μικρότερες είτε ίσες από τις υπολειπόμενες διαθέσιμες διαστάσεις του κάδου. Χρήση νέου κάδου πραγματοποιείται όταν δεν χωράει κάποιο από τα υπολειπόμενα αντικείμενα. Σκοπός του προβλήματος είναι η χρήση όσο το δυνατόν λιγότερων κάδων για την αποθήκευση όλων των αντικειμένων.

Δημιουργείστε ένα πρόγραμμα το οποίο θα αντιστοιχίζει αντικείμενα βάρους έως 20 κιλών σε ομοιογενείς αποθηκευτικούς χώρους που αντέχουν μέχρι 20 κιλά. Το πρόγραμμα θα δέχεται έως και 10 αντικείμενα. Υποθέστε ότι έχουμε άπειρο αριθμό διαθέσιμων κάδων. Χρησιμοποιείστε για την επιλογή του επόμενου αντικειμένου της λύσης τα κριτήριο που περιγράφονται στη συνέχεια. Εκτυπώστε τη λύση.

Κριτήριο Επιλογής Επόμενου Στοιχείου
Αφού ιεραρχηθούν τα αντικείμενα βάσει βάρους επιλέγεται το πρώτο διαθέσιμο και στη συνέχεια το επόμενο θα είναι όποιο αντικείμενο αφήνει ελάχιστο υπολειπόμενο βάρος. Παράδειγμα λύσης ακολουθεί:

```
Insert package's weight or -1 to exit
10
Insert package's weight or -1 to exit
20
Insert package's weight or -1 to exit
15
Insert package's weight or -1 to exit
4
Insert package's weight or -1 to exit
2
Insert package's weight or -1 to exit
3
Insert package's weight or -1 to exit
11
Insert package's weight or -1 to exit
9
Insert package's weight or -1 to exit
5
```

```
Insert package's weight or -1 to exit
6
Bin=1 weight=20
Bin=2 weight=15
Bin=2 weight=5
Bin=3 weight=11
Bin=3 weight=9
Bin=4 weight=10
Bin=4 weight=6
Bin=4 weight=4
Bin=5 weight=3
```

(Exercise difficulty : Hard)

.

References

https://projecteuler.net/
https://www.hackerrank.com/
Java: How to Program, by Harvey M. Deitel, 6th Edition.