QA Consulting

JavaSE Exercise Book

TRAINING MATERIALS

Contacts

elliot.womack@qa.com

team.qac.all.trainers@qa.com

www.consulting.qa.com

CONTENTS

Brief 3

Submission 3

Exercises - Level 1 Basic 4

Hello World! — 4

Assignment — 4

Parameters — 4

Return types — 4

Parameters/Operators — 4

Conditionals — 5

Conditionals 2 — 5

Iteration — 5

Arrays — 5

Iteration/Arrays — 6

Iteration/Arrays 2 — 6

Exercises - Level 2 Intermediate 7

Blackjack — 7

Unique Sum — 7

Too hot? — 7

People — 8

Garage — 8

Paint Wizard — 9

Working with Files — 9

Junit — 10

Library/TDD — 10

Exercises - Level 3 Advanced 13

HashMaps – Anagrams — 13

Prime Numbers 1 — 13

Prime Numbers 2 — 13

Strings — 14

Strings 2 — 14

Battleships — 14

Brief

This is the exercise book for the entire Java SE course, this isn't to say this will be all the work you will be doing during the course, additional exercises will be handed out as the week goes.

As soon as all basic tasks are completed, before moving on notify your trainer to verify that you have completed them correctly.

Advanced exercises are for those that are quite ahead and wish to challenge themselves, however this does not mean you may skip exercises unless the trainer gives permission to do so.

If a method gives an example in the format:

```
method(10,10,)-> 5
```

`method` – This is the name of the method, this should be appropriate to the exercise you are doing.
`(10,10)` – These are the parameters to that method, aka the data that the method accepts.
`-> 5` – This is the return value of the method, done via a return statement.
This is not to say if the exercise does not provide an example that you should not create methods like these.

Submission

Exercises will be submitted via Github, an appropriate folder structure is expected.

Exercises - Level 1 Basic

HELLO WORLD!

Output “Hello World!” to the console via a **`System.out.println()`** statement in your main method.

ASSIGNMENT

Store “Hello World!” in a variable, then output it to the console via a **`System.out.println()`** statement in your main method.

PARAMETERS

Create a method that accepts a string as a parameter, and then outputs that string to the console via a **`System.out.println()`**, then call that method from your main method.

RETURN TYPES

Create a method to return “Hello World!” once called, which you call from your main method, to then output the text to the screen.

PARAMETERS/OPERATORS

Create a method that accepts two integers as parameters, then returns an integer that is a sum of the two integers given, then call this method from your main method and output the returned result.

CONDITIONALS

Modify your method from the previous task to accept another parameter, a Boolean, which if it is true, the method will return a sum of the two numbers, and if it is false it will return the multiplication of the two numbers.

EXAMPLE

```
method(2, 3, true) -> 5
method(3, 3, false) -> 9
```

CONDITIONALS 2

Modify your method from the previous task to have another if statement that checks if one of the numbers is 0, if this is true then return the other non-0 number.

EXAMPLE

```
method(1, 0) -> 1
method(1, 2) -> 3
```

ITERATION

Create a for loop that will call and output the result of your method from Conditionals 2 10 times, using the current iteration as one of the parameters you pass to it.

ARRAYS

Create an array that will hold 10 integer values, populate the array with values, then call and output the result of your method from Conditionals 2, passing values that are stored in the array as arguments to the method.

ITERATION/ARRAYS

Using your array that you created in Task 9, create a for loop that iterates through your array, outputting the values contained within it.

ITERATION/ARRAYS 2

Create a for loop that populates an integer array with values, outputting them at each iteration. Then create another loop that iterates through the array, changing the values at each point to equal itself times 10, outputting them at each iteration.

EXAMPLE OUTPUT

```
1,2,3,4...
10,20,30,40...
```

Exercises - Level 2 Intermediate

BLACKJACK

Given 2 integer values greater than 0, return whichever value is closest to 21 without going over 21. If they both go over 21 then return 0

EXAMPLE

```
method(18,21) -> 21
method(20,18) -> 20
method(22,22) -> 0
```

UNIQUE SUM

Given 3 integer values, return their sum. If one value is the same as another value, they do not count towards the sum. Aka only return the sum of unique numbers given.

EXAMPLE

```
method(1,2,3) -> 6
method(3,3,3) -> 0
method(1,1,2) -> 2
```

TOO HOT?

Given an integer value and a Boolean value, temperature and isSummer, if temperature is between 60 and 90 (inclusive), unless its summer where the upper limit is 100 instead of 90. Return true if the temperature falls within the range, false otherwise.

PEOPLE

Create a Person class that models the following:

- Name
- Age
- Job Title

And has a method to return all three of these in a formatted string. (Override the toString() method!)

Create some example objects with this class.

Create an ArrayList and store those objects inside.

Use an enhanced for loop to output all of your people to the console.

Create a method that searches for the Person object by their name.

GARAGE

Using Vehicle as a base class, create three derived classes (car, motorcycle etc.), each derived class should have its own individual attribute in addition to the normal Vehicle attributes that it inherits.

Using a List implementation store all your vehicles in a Garage class (e.g. ArrayList) Create a method in Garage that iterates through each Vehicle, calculating a bill for each type of Vehicle in a different way, depending on the type of vehicle it is.

Garage should have methods that add Vehicle, remove Vehicle(s) (By ID, By Vehicle Type) fix Vehicle (Calculate bill) and empty the garage.

PAINT WIZARD

Create a paint requirement wizard that will calculate which of the following three products, would be the cheapest to buy, for the room you are painting.

1) CheapoMax (20Litre) £19.99

- This tin of paint will cover 10m2 per Litre

2) AverageJoes (15 Litre) £17.99

- This tin of paint will cover 11m2 per Litre

3) DuluxourousPaints (10 Litre) £25

- This tin of paint will cover 20m2 per Litre

Work out which one wastes the least.

Work out which is the cheapest paint brand to buy for any room.

WORKING WITH FILES

Create a class representing a person with 3 attributes Name, Occupation, Age
Create an array list and populate it with 5 of these objects (Make up the values etc.)

Create a loop to iterate through the ArrayList, writing each object to one file, recommended IO library is BufferedWriter (Think about how you format this)

Separately, create another ArrayList and populate it with the data from the file you just created. Recommended IO library is BufferedReader (You're going to have to parse it back in the format you wrote it in, use `String.split())`

JUNIT

Implement unit tests for tasks 13,14,15.

Simulate normal inputs and test their outputs

*You may notice your methods were not working with inputs (parameters) and outputs (return statements), instead using scanners and system.out.println*s. *this showcases a* ***bad*** *method. If it's not testable, it's not good. Consider re-creating those methods.*

LIBRARY/TDD

Create a library system with functionality to manage items within the library.

EXPECTATIONS

Class diagram created and adhered to as much as possible.

TDD Implemented.

At least one Abstract Class must be implemented.

At least one Interface Class must be implemented.

Each item should have at least 1 additional attributes unique to itself.

Method Overloading implemented.

Method Overriding implemented.

Correct usage of access modifiers

Naming conventions adhered too

Commenting when appropriate. (Complex methods)

ITEMS

Three of the following:

- Books
- Maps
- Government documents
- Media resources (Camera etc.)
- Newspapers
- Journals
- Magazines
- Dissertations
- Theses

FUNCTIONS

All of the following:

- Check out item
- Check in item
- Add item
- Remove item
- Update item
- Register person
- Delete person
- Update person

STATIC

Implement an ID system in your library project, utilising a static integer variable.

IO

Create a method to write the current library contents (Any items currently in the library) to a file, and another method to load them from a file into the library.

ADDITIONAL

Create a command line interface (CLI) for your library.

This should be a separate class to your Library system, do not change your code to integrate a CLI into it, it should be able to just fit around it.

Exercises - Level 3 Advanced

HASHMAPS – ANAGRAMS

- Create a method that reads words from a file, one per line, and stores them in an Array.
- Create a method that takes a String as a parameter, and returns a sorted version of the string back. (bac)->abc

Using these methods and a HashMap, perform the following;

- Find the word that has the most anagrams located in the list
- If two words have the same amount of anagrams, output the word that is longer.
- If two words have the same length and the same amount of anagrams, output both.

Email your lecturer to get access to the proper wordlist once your code is ready. Until then create your own text file to test it on.

Hint – The key will be the sorted word, the value will be an arraylist of the anagrams found.

PRIME NUMBERS 1

Create an algorithm that determines how many prime numbers are between 0 and 3 million.

Additional *– Have it finish running in under 2 minutes*

PRIME NUMBERS 2

Create an algorithm that determines how many prime numbers are between 0 and 2 billion

.

Additional *– Have it finish running in under 3 minutes*

Additional *– Have it work from 0 - 3 Billion*

STRINGS

Given two strings, write a program that efficiently finds the longest common subsequence.

STRINGS 2

Given two strings, write a program that outputs the shortest sequence of character insertions and deletions that turn one string into the other.

BATTLESHIPS

Create the battleships game!

This project is to create a digital version of the popular board game known as battleships. Battleships is a two player game with 2 phases. In the first phase the player's ships are placed on the board. In the second phase each player takes it in turns to select grid squares on the board in an attempt to find and destroy their opponent's ships. Once one player has lost all of their ships the game is over and the player who still has ships on the board is the winner.

Each player has a number of ships including: 2 patrol boats (1 x 2), 2 battleships (1 x 3), 1 submarine (1 x 3), 1 destroyer (1 x 4) and 1 carrier (1 x 5).

There are a number of rules that players must follow.

- 2 Ships cannot occupy the same space on the board.
- If a player scores a 'hit' on their opponent, then they get a another shot.
- Ships cannot be moved once they have been placed.

Advice

Battleships is a seemingly simple strategy game but without careful planning it can be easy to become "lost" in the project. It is recommended that you attempt to complete the project in a set of stages where with each stage you increase the level of complexity. Remember that as the complexity of your project increases you may find that you wish to go back to a previous version in some cases so it is highly advised that you create versioned copies of your project at each stage. An advised set of stages are:

Stage 1:

A 3 x 3 grid with one ship that is 2 pieces long is placed on in the grid.

Stage 2:

A 3 x 3 grid with 2 ships that are 2 pieces long and placed on the grid with validation to ensure legal placement.

Stage 3:

Two 3 x 3 grids with 2 ships where players take alternating turns taking shots at the other grid.

Stage 4:

Differentiation between 'hits' and 'misses' implemented.

Stage 5:

Checks for sunk ships with game over when one player has lost all their ships.

Stage 6:

Two 12 x 12 grids with all 7 ships placed in valid locations.

Stage 7:

Players can select the placement of their ships on the grid during phase 1.

Stage 8:

Implementation of an AI player.

Stage 9:

Players can only see their own grid with shots taken.

Stage 10:

Implementation of a GUI (Swing/JavaFX)