|  |
| --- |
| Macintosh HD:Users:hrente:Desktop:QA_branding:Logo:QA Consulting Logo_solo-01.jpg |
| Lecture Exercise  Booklet |
| Java Standard Edition |

Contents

[Basic 1](#_Toc474835922)

[Hello World! 1](#_Toc474835923)

[Assignment 1](#_Toc474835924)

[Parameters 1](#_Toc474835925)

[Return types 1](#_Toc474835926)

[Parameters/Operators 1](#_Toc474835927)

[Conditionals 1](#_Toc474835928)

[Conditionals 2 1](#_Toc474835929)

[Recursion 1](#_Toc474835930)

[Arrays 1](#_Toc474835931)

[Recursion/Arrays 1](#_Toc474835932)

[Recursion/Arrays 1](#_Toc474835933)

[Scanner 1](#_Toc474835934)

[Intermediate 1](#_Toc474835935)

[Blackjack 1](#_Toc474835936)

[Unique Sum 1](#_Toc474835937)

[Too hot? 1](#_Toc474835938)

[People 2](#_Toc474835939)

[Garage 2](#_Toc474835940)

[Paint Wizard 2](#_Toc474835941)

[Junit 2](#_Toc474835942)

[Working with Files 3](#_Toc474835943)

[Library/TDD 3](#_Toc474835944)

[Static 4](#_Toc474835945)

[IO 4](#_Toc474835946)

[Advanced 5](#_Toc474835947)

[HashMaps – Anagrams 5](#_Toc474835948)

[Prime Numbers 1 5](#_Toc474835949)

[Prime Numbers 2 5](#_Toc474835950)

[Strings 5](#_Toc474835951)

[Strings 2 5](#_Toc474835952)

[Battleships 5](#_Toc474835953)

# Basic

### Hello World!

Output “Hello World!” to the console.

### Assignment

Store “Hello World!” in a variable, then output it to the console.

### Parameters

Create a method that accepts a string as a parameter, and then outputs that string to the console.

### Return types

Modify your method to return a string once called, which you then use to output to the screen.

### Parameters/Operators

Create a method that accepts two integers as an input, then returns an integer that is a sum of the two integers given.

### Conditionals

Modify your method from Task 5 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.

### Conditionals 2

Modify your method from Task 6 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.

**Input -> 1, 0 Return 1**

**Input -> 1, 2 Return 3**

### Recursion

Create a **for** loop that will call and output the result of your method from Task 7 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 Task 7 with values that are stored in the array.

### Recursion/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.

### Recursion/Arrays

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…

### Scanner

Modify the previous task to use the Scanner class, taking an integer off of the user and using that integer to determine how large the array is going to be.

# 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

**Input(18,21) -> 21**

**Input(20,18) -> 20**

**Input(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.

**Input(1,2,3) -> 6**

**Input (3,3,3) -> 0**

**Input (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.

Create some example objects with this class.

Create an ArrayList and store those object inside.

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

### Garage

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

Using a **List** implementation store all your vehicles in a **Garage** class

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.

Work out which one wastes the least.

Work out which is the best choice for any room (Cheapest).

**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**

### Junit

Implement unit tests for tasks **13,14,15**.

Simulate normal inputs and test their outputs

### 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 (Think about how you format this)

Separately, create another ArrayList and populate it with the data in the file you just wrote too. (You’re going to have to parse it back In the format you wrote it in)

### 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.

Correct usage of access modifiers

Naming conventions adhered too

Commenting where necessary.

**Items:**

At least 3 of the following:

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

**Functions:**

At least 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.

# 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 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 1 and 3 million.

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

### Prime Numbers 2

Create an algorithm that determines how many prime numbers are between 1 and 2 **billion**.

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

### 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 second 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 has been attempted.