# **COMP 1510 Notes Set 1A**

## Preface

Please keep the notes consistent with Java structure

Aka // or /\* \*/ around any comments, and /\*\* \*\*/ goes before the code. Thanks!

# Terminology

Volatile // memory that you lose if the power goes out. Ex. main memory

Instantiation // The act of creating an object from a class.

String object is immutable, but with a string method it can return a new string object that has been mutated.

## Static

* Static variables are the same instance for all classes
* Static methods can be invoked through their class name without an object (ex. math.sqrt(27))
* A static method can reference static variables

## Final

Final keyword has a numerous way to use:

A final class cannot be subclassed.

A final method cannot be overridden by subclasses

A final variable can only be initialised once

Other usage:

When an anonymous inner class is defined within the body of a method, all variables declared final in the scope of that method are accessible from within the inner class

A static class variable will exist from the start of the JVM, and should be initialised in the class. The error message won't appear if you do this.

## Decomposition

Breaking a complex method down into smaller parts;

# Formulas

The difference between .next and .nextLine is:

nextLine takes the entire line, next only takes the next input

### String formats

stringVariable.length(); // provide the length of the string variable

stringVariable.charAt(1)); // character at the index 1

stringVariable.toUpperCase();

stringVariable.substring(0, 20); // takes the substring from 0-20 (inclusive of 0, 20 will be exclusive)

stringVariable.replace(‘B’, ‘M’) // replaces B with M

stringVariable.concat(stringVariable2); // concatenates two strings together

stringVariable.equalsIgnoreCase(String str); /\* Returns true if this string contains the same characters as a str (without regard to case) and false otherwise. \*/

### Enumerated Types

enum Season {winter, spring, summer, fall}

Season time = Season.spring;

### Scanner format

/\* If you are planning to read a scan.nextLine after scan.nextInt, make sure to place an empty scan.nextLine() in between. \*/

Import java.util.Scanner; + scannerName.close();

## Scanning files

Import java.io\*

Scanner fileScanner = new Scanner(new File(“filename”));

String lineScanner = fileScanner.nextLine();

Scanner urlScan = new Scanner (lineScanner);

urlScan.useDelimiter(“/”) // separates by “/”, useDelimiter is a Scanner method, need to invoke on a scanner to work

### Number format

Import java.text.NumberFormat;

NumberFormat percent = NumberFormat.*getPercentInstance*();

percent.setMinimumFractionDigits(2);

percent.format()

### DecimalFormat

/\*\*

\* Same as NumberFormat but with more control over the number of digits and leading zero.

\*\*/

java.text.DecimalFormat;

DecimalFormat formatName = new DecimalFormat(“#0.000”);

formatName.format(0.12045); // yields “0.120”

### How to use class constructor:

Public Class {

Private int theVariables;

Public Class(int i) {

theVariables = i; //initialise the base stats.

}

Public double actions() {

return actions;

}

}

### Math Class

/\*\*

\* The static function Math.min() returns the lowest-valued number passed into it, or NaN if any parameter isn't a number and can't be converted into one.

\*/

Math.min(2, 3, 1);

Math.PI();

*/\*\**

*\* a= the number*

*\* b= exponents*

*\**

*\* E.g : 2 ^ 8*

*\**

*\*/*

Math.pow( a, b)

Math.pow (2, 8)

Math.random () // gives a number between 0(inclusive) , 1(exclusive)

// Use multiplication to input the range you are looking for.

// + Or - gives you the min range you are looking for.

(Math.random() \* range) + minimum; // for random number within a range

Math.abs(-10) // gives absolute value 10

### java.util.Random;

randomGenerator.nextfloat(); // returns a value 0 or 1

# Arrays

## Different array formats

import java.util.ArrayList;

ArrayList <Integer> “arrayName” = new Arraylist<Integer>();

arrayName.add(5);

arrayName.remove(5);

arrayName.size();

arrayName.get(5);

arrayName.set(5, 2); // sets index 5 item to value 2

arrayName.length;

Import java.util.Collections;

Collections.sort(arrayName)

/\*To loop through, reads “for each string “i” in arrayName”. “I” becomes the index \*/

for (String i : arrayName) {

“arrayName” [] = {item1, item2, item3}

## Double array

<https://math.hws.edu/javanotes/c7/s5.html#:~:text=A%202D%20array%20has%20a,and%20the%20number%20of%20columns>.

int [][] “arrayName” = new “arrayName” [numberOfRows][numberOfColumns];

or

int[][] A = {

{ 1, 0, 12, -1 },

{ 7, -3, 2, 5 },

{ -5, -2, 2, -9 }

};

arrayName[row3][column4] = 8; // changes the -9 above to 8

enumeratedItem.ordinal() // returns the number of the item in enumerated list

## Insertion Sort (ON^2)

// always shows the first line

<https://www.softwaretestinghelp.com/insertion-sort-in-java/#:~:text=Answer%3A%20Insertion%20sort%20is%20a,placed%20in%20its%20proper%20position>.

## Selection Sort (ON^2)

// always shows the first line

<https://www.geeksforgeeks.org/selection-sort/>

# How to Use Interfaces:

<https://www.youtube.com/watch?v=kTpp5n_CppQ&ab_channel=AlexLee>

Main use: so you don’t have to keep adding the same attributes to multiple classes. Gives you a framework of methods/variables.

## Constructors:

public interface “interfaceName”}

//interfaces have attributes and things it can do. These can also be classes

public void exampleClass(int attribute);

public int class(); //public by default

public final int numberConstant;

}

//in the class using the interface, add “implements” “interfaceName”. Example:

public class Question implements Complexity {

// must provide a definition for all methods in the interface. Example:

public void exampleClass (int x) {

Int i = x/69;  
 }

public int class() {

return y;

}

}

## Driver side:

Question q1;

String userInput;

Scanner Scan = new Scanner(System.in);

q1 = new Question(“wtf”, “comp1510”);

q1.setComplexity(99);

System.out.println(“Level: “ + q1.getComplexity()”);

# Useful Shortcuts

Ctrl + shift + S, then R → automatically create getters and setters

Place your cursor on the method's name inside the test class. ...

Press Alt+Shift+X,T to run the test (or right-click, Run As > JUnit Test).

# Types of Errors

Logical error

* Shit, I fucked up - logical error, gives the wrong result.
  + Example: gives the wrong output. Maybe you wanted an else statement to appear but the code never makes it to the else statement, giving an unintended result.

Compile-time error

* Shit, it didn’t work
  + Examples: “ “ “ doesn’t know what to do with 3rd quotation. Enumerated list where u try to call something that isn’t in the list. Saying a function that doesn’t exist yet

Syntax error (subset of compile-time error)

* Shit, somethings wrong
  + Examples: // syntax error caused by right side providing a double and the left side wants int

Run-time error

* Shit, it crashed
  + Examples: you divided by zero