

CS2040 Lab 1

Java Introduction

Lab Introduction

- 2-hour session every Friday
- Intended to provide hands-on experience with programming
- Involves one-day assignments, and take-home assignments
 - One-day assignments (solving 1 problem) should ideally be completed within the lab itself. The actual duration of the deadline is **10am on Friday to 10am on Saturday**, but the assignment is “*doable*” within the duration of the lab.
 - Take-home assignments (solving 2 problems) should be completed before the deadline (2 weeks per take-home assignment). Released on **Tuesdays 10am**.
- Slides will be accessible in Canvas under “Files > Labs” at 6pm on Fridays, along with some hint slides for the one-day assignment

Lab Structure

- 1st part of labs (max. 30 min): short lesson
 - Cover answers for the previous session's assignments (one-day, and take-home, where applicable)
 - Cover relevant Java API for the session's given topic
 - Brief discussion of the one-day assignment
- 2nd part of labs: solving the one-day assignment (graded)
 - The first 20-30 minutes of this part will be used for students to plan how to solve the assignment, and express it in terms of pseudocode
 - Allowed to discuss at algorithm level with other students, but no discussing/sharing of code
 - **Type / write out the pseudo-code for your solution (should be done individually) which must be submitted to the TAs**
 - Submission of the pseudo-code will count as your attendance for that week's lab
 - Can continue working on the assignment after the lab if necessary

Kattis Introduction

- Online platform used for submitting and grading assignments
- Found at <https://nus.kattis.com/>

*Everyone should have already signed up. If you have not please sign up, refer to lecture notes 0 on how to sign up (the registration key should have been sent to your canvas mail box).

Lab 1 – Runtime Analysis

- In CS2040, sometimes having a program/algorithm give the correct answer may not be sufficient; it may be required that your program/algorithm should run quickly as well
- Kattis provides the maximum running time for a program in the problem description on the right, under “CPU Time limit”
- Generally, machines on Kattis can run close to 100 million operations a second
- Try calculating whether your program can pass the time limit based off its time complexity
- Eg for a problem where $N \leq 1000$, and the time limit is 1 sec, $O(N^3) = 1$ billion = probably not okay (> 100 million), whereas $O(N^2) = 1$ million = okay

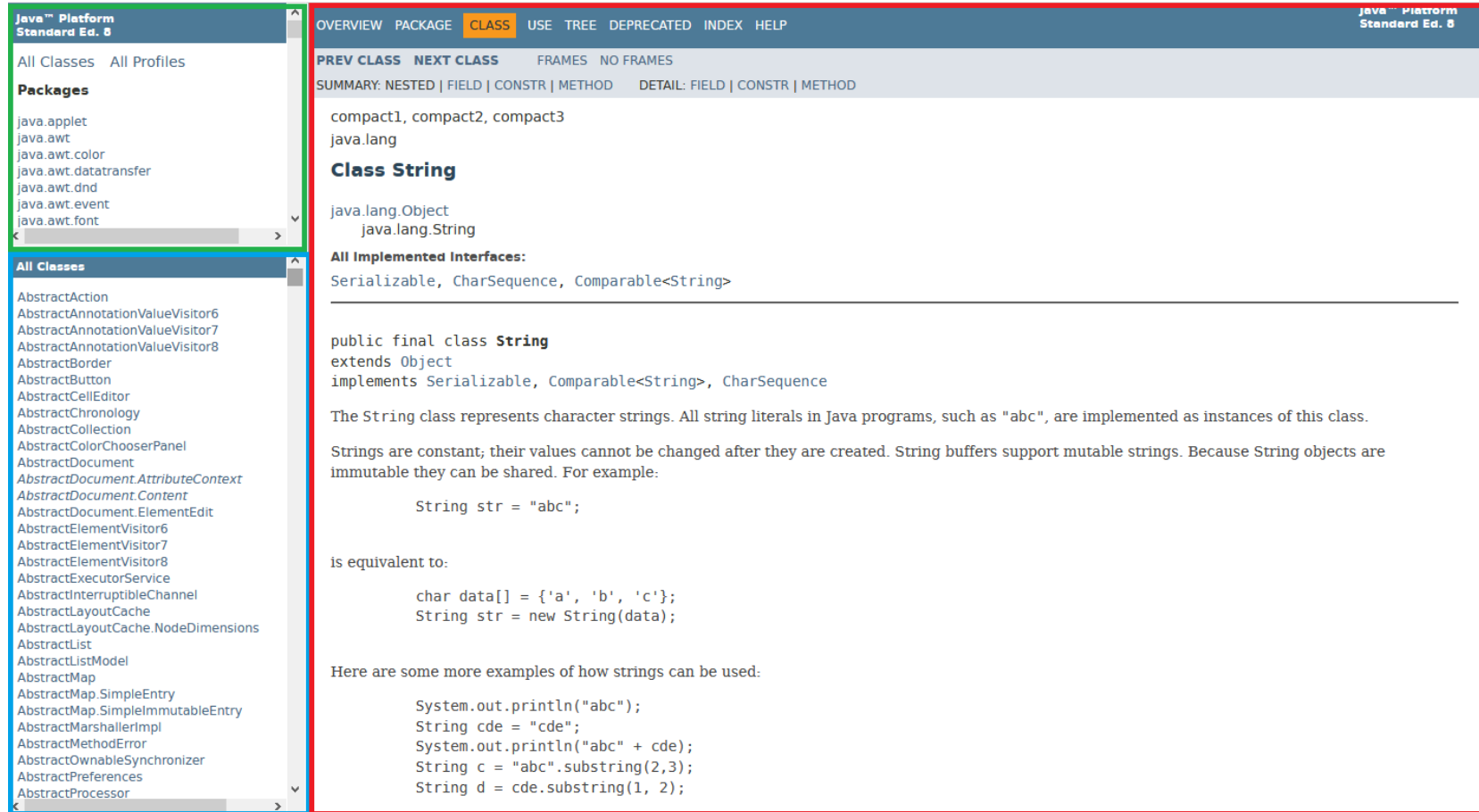
Lab 1 – Runtime Analysis

- Also note that the methods provided by Java API may not run in $O(1)$ time, and as such your program may run slower than intended if this is not taken into consideration
- To fully understand the methods provided by the API, it is recommended to read up the documentation
- The following slides are based off the Java 8 API documentation:
<https://docs.oracle.com/javase/8/docs/api/>
 - Documentation of the later versions of Java (up to Java 10) follows the same idea, but uses a different layout by default. To view them in the same manner, click on “FRAMES” at the top of the page
 - Documentation from Java 11 onwards does not seem to support frames, requiring manual navigation of the packages (or just search for the class name via Google, or the search bar at the upper right of the Java API page)

Lab 1 – Reading Java API

Package window: can click on them to view only classes from a single package

Class window: click on one of them to view the full API of that class



Main window: shows all the details of the selected class

Lab 1 – Reading Java API

- For the package window, clicking on a package will switch the class window to show only the classes in that package
 - Frequently used packages are:
 - `java.util`
 - Almost everything that's covered later in the module
 - `java.lang`
 - Strings and related classes
 - Wrapper classes
 - `java.io`
 - Buffered input/output (covered in a later lab)
 - For Java 9 onwards, these are found under the `java.base` package in the documentation (but retain the same package names otherwise)

Lab 1 – Reading Java API

- For the main window, the contents tend to be sorted as follows:
 - 1. Brief description of class
 - 2. Fields, constructors and methods (short version, sorted alphabetically)
 - Click on an entry here to go to the relevant entry in section 3
 - 3. Fields, constructors and methods (long version, may not be sorted)
- The time complexity of a method will sometimes be stated in section 1 or 3

Lab 1 – Useful API

- Scanner class – used for reading input
- Found in the java.util package; need to use the following line to import:
 - `import java.util.*;`
- Declare a new Scanner object with the following line (in main method):
 - `Scanner sc = new Scanner(System.in);`
 - Do not create more than 1 Scanner object in your program, as using multiple Scanner objects can cause problems
- Read in input using the methods found in Scanner:
 - `int testCases = sc.nextInt();`

Lab 1 – Scanner

Method name	Description	Time
<code>.nextInt()</code>	Reads the next token in the input as an integer	$O(n)$
<code>.nextDouble()</code>	Reads the next token in the input as a double	$O(n)$
<code>.next()</code>	Reads the next token in the input as a String	$O(n)$
<code>.nextLine()</code>	Reads until it reaches the end of the line	$O(n)$

n refers to the length of the input that is read

These slides covering API will cover the most frequently used (but not all) methods of a class; for a full list, refer to the Java API documentation

Lab 1 – System.out (not really a class)

Method name	Description	Time
<code>System.out.print(String str)</code>	Prints <i>str</i>	$O(n)$
<code>System.out.println(String str)</code>	Prints <i>str</i> , followed by a newline character (<code>'\n'</code>)	$O(n)$
<code>System.out.printf(String format, Object... args)</code>	Emulates the printf function from C	$O(n)$

n is the length of the output

Technical note: `System.out` is an instance of the `PrintStream` class, so you may refer to the API documentation on `PrintStream` (in `java.io`) to explore more methods

Lab 1 – Useful API

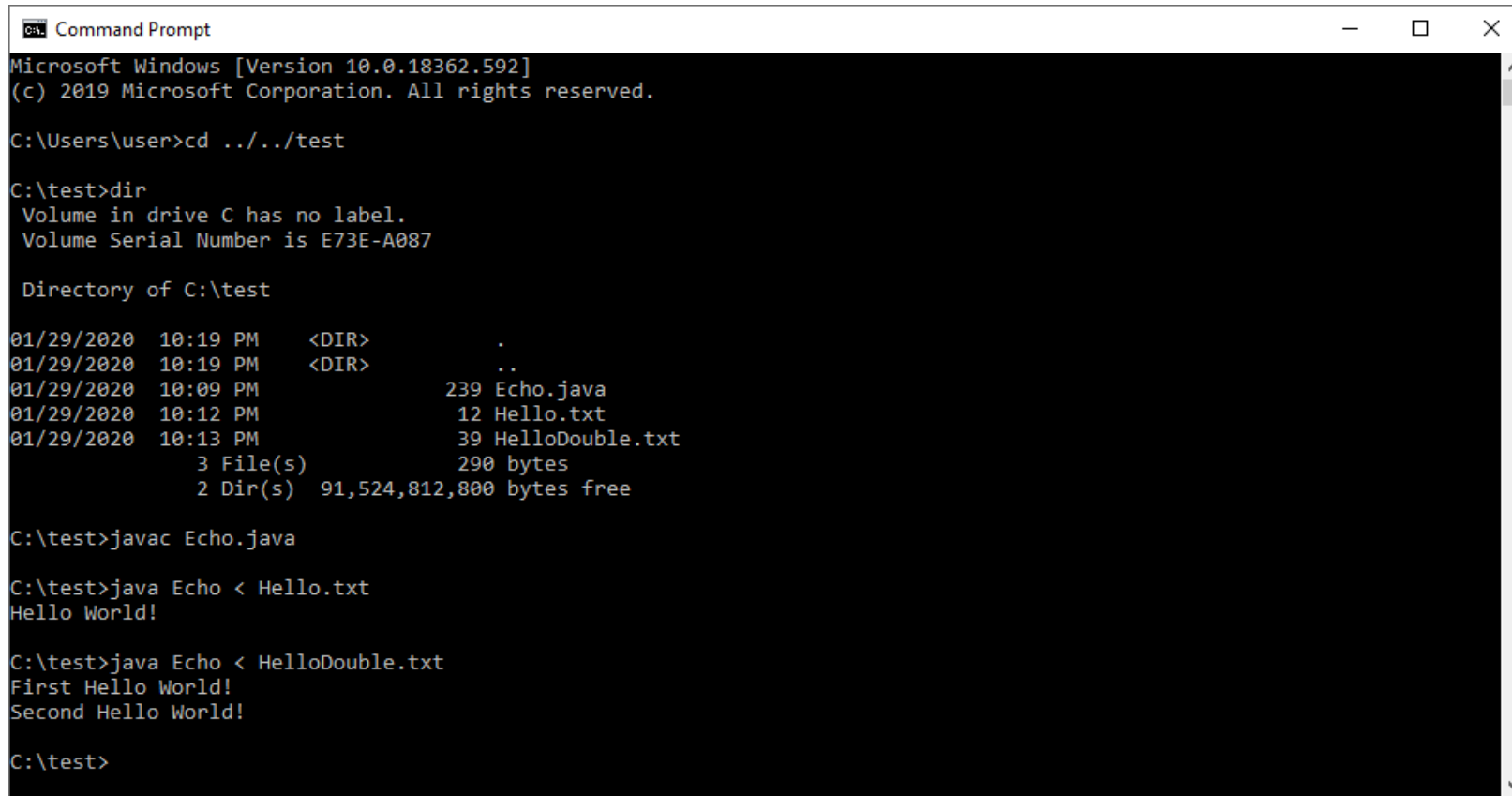
- String class – contains several methods that could be of use
- Found in the java.lang package; is imported by default
- Can be constructed in various ways:
 - `String str1 = new String("apple");`
 - `String str2 = "apple";`
 - `char[] arr = {'a', 'p', 'p', 'l', 'e'};`
 - `String str3 = new String(arr);`

Lab 1 – String

Method name	Description	Time
<code>.split(String delim)</code>	Splits a String into an array of Strings, based off the characters found in <i>delim</i>	O(length of string)
<code>.charAt(index i)</code>	Returns the character at index <i>i</i> (0-based)	O(1)
<code>.equals(String other)</code>	Checks if this string has the same value as the value of String <i>other</i> Note: not the same as using <code>==</code>	O(length of shorter string)
<code>.concat(String other)</code>	Returns a new string which is this string + <i>other</i> Note: does not modify the original String (Strings are immutable in Java)	O(length of resulting string)
<code>.length()</code>	Returns the length of the string	O(1)
<code>.substring(int start, int end)</code>	Returns a new string, which contains the content of the original string from index <i>start</i> (inclusive) to index <i>end</i> (exclusive) (indices are 0-based)	O(length of resulting string)

Lab 1 – Command Line/Terminal Usage

- It is helpful to know how to compile and execute Java programs via command line (example below)



```
Command Prompt
Microsoft Windows [Version 10.0.18362.592]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd ../../test

C:\test>dir
Volume in drive C has no label.
Volume Serial Number is E73E-A087

Directory of C:\test

01/29/2020  10:19 PM    <DIR>          .
01/29/2020  10:19 PM    <DIR>          ..
01/29/2020  10:09 PM                239 Echo.java
01/29/2020  10:12 PM                 12 Hello.txt
01/29/2020  10:13 PM                 39 HelloDouble.txt
               3 File(s)                290 bytes
               2 Dir(s)  91,524,812,800 bytes free

C:\test>javac Echo.java

C:\test>java Echo < Hello.txt
Hello World!

C:\test>java Echo < HelloDouble.txt
First Hello World!
Second Hello World!

C:\test>
```

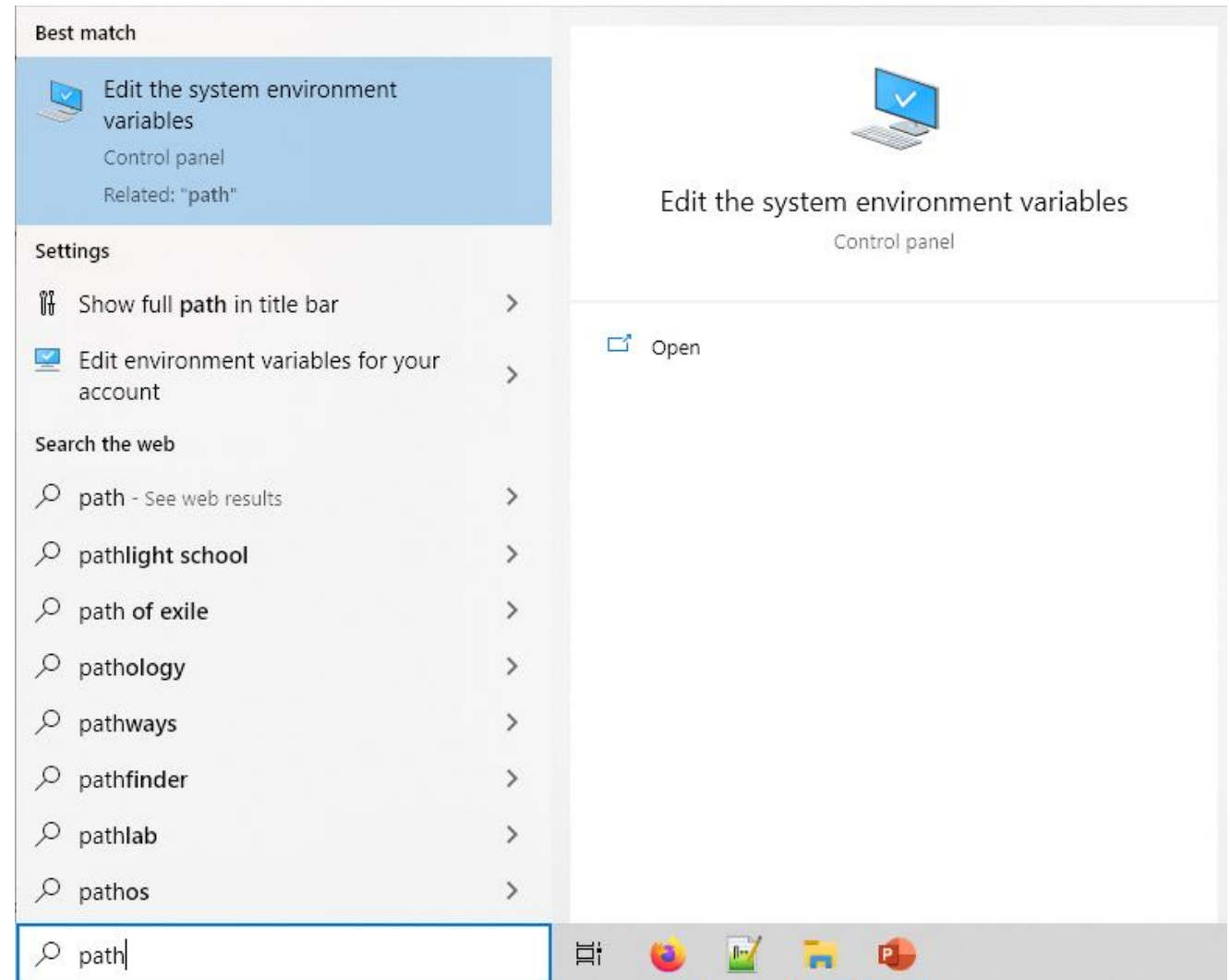
Lab 1 – Command Line/Terminal Usage

- Using the command line requires setting up the location of your java.exe/javac.exe files in your PATH environment variable first, otherwise the command line will be unable to recognise "javac" and "java" as valid commands
- The following slides can also be found as a standalone document on Canvas in the Lectures > Java Intro and ADT folder

Editing PATH Environment Variable

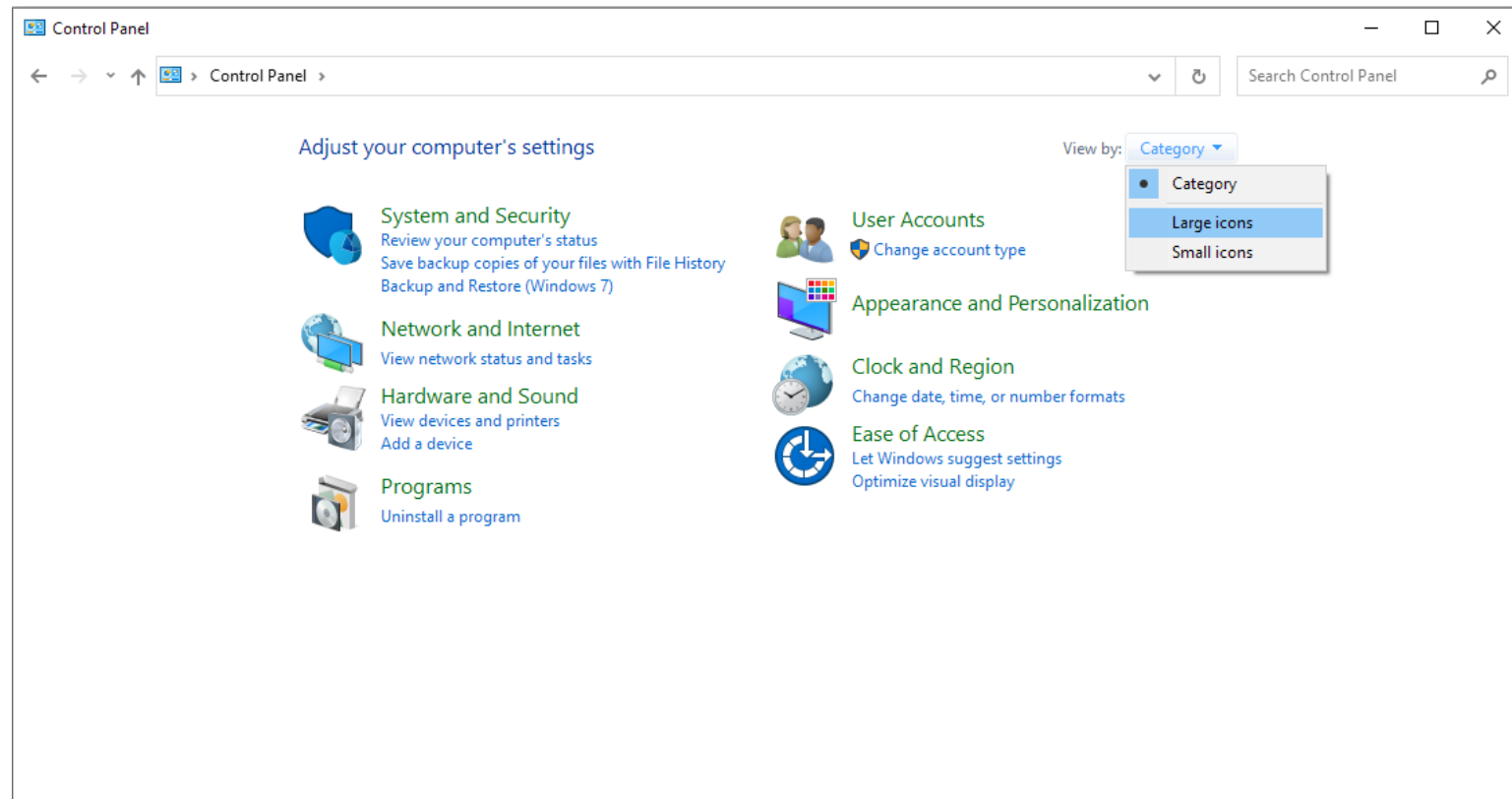
If you have the Windows search bar active, then accessing environment variables can be easily done by searching for "path" and clicking on the first search result

Skip to slide 20 if you do this



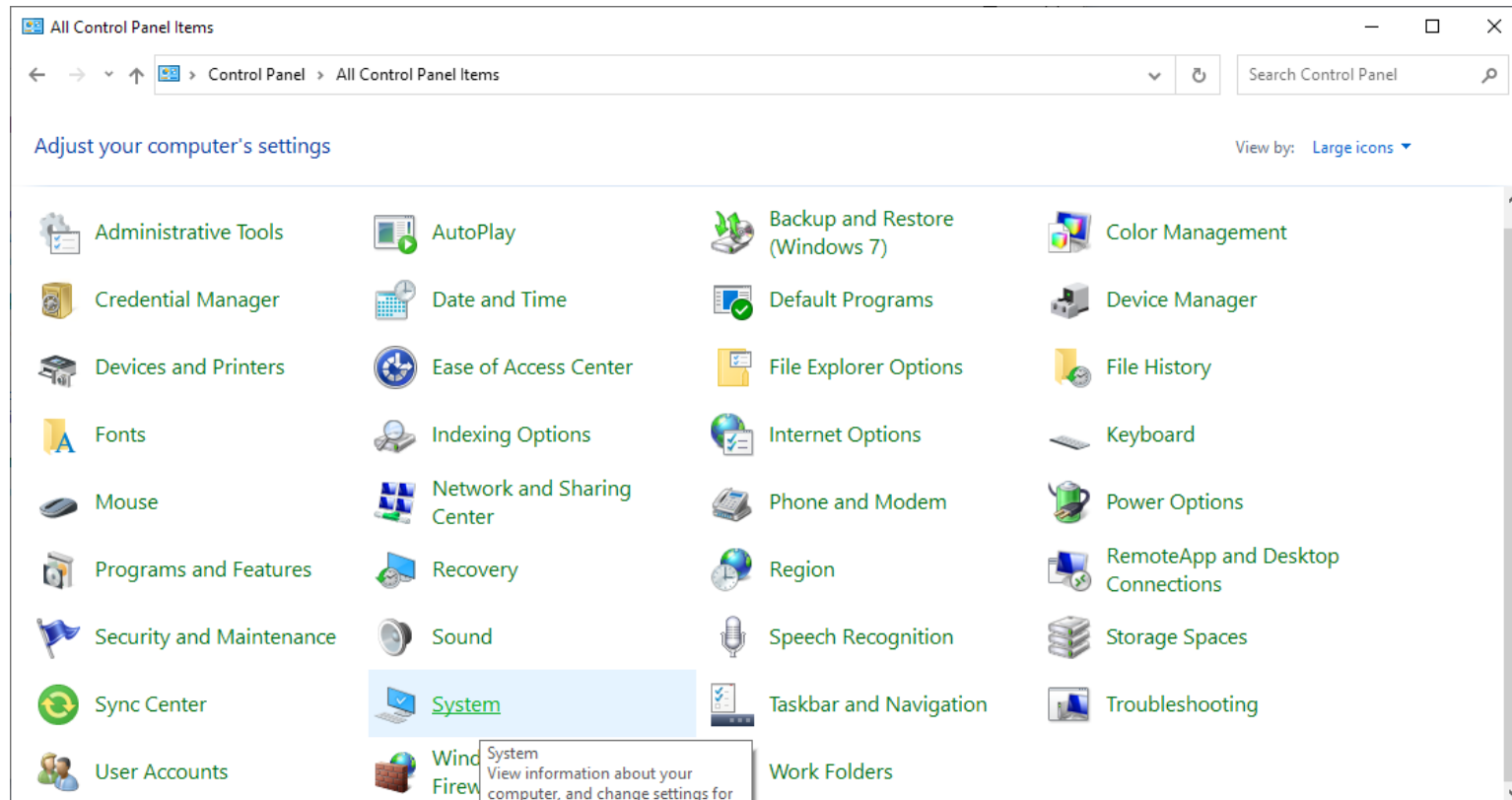
Editing PATH Environment Variable

Otherwise, open the control panel and set "View by:" to "Large icons"



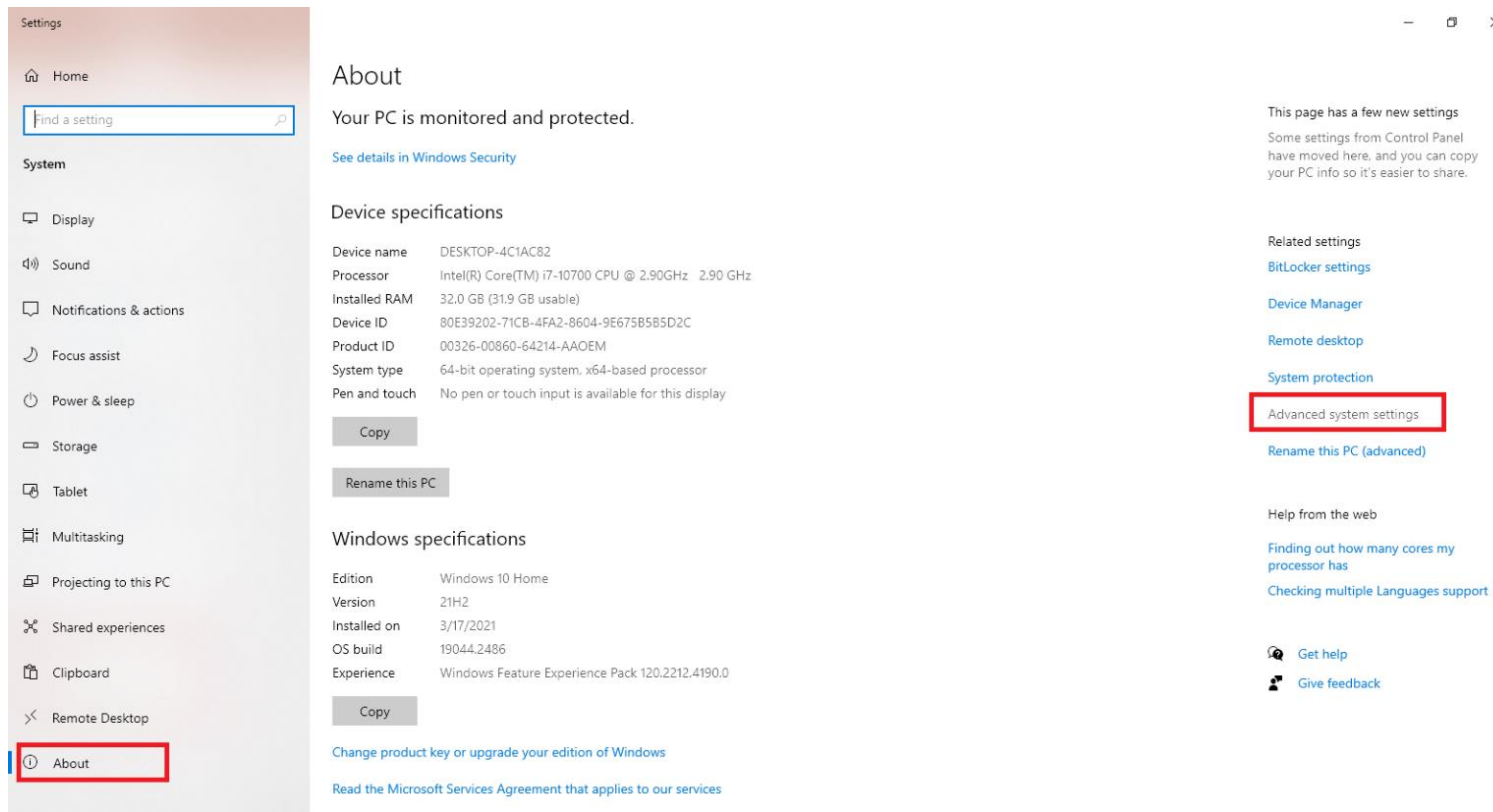
Editing PATH Environment Variable

Click on "System"



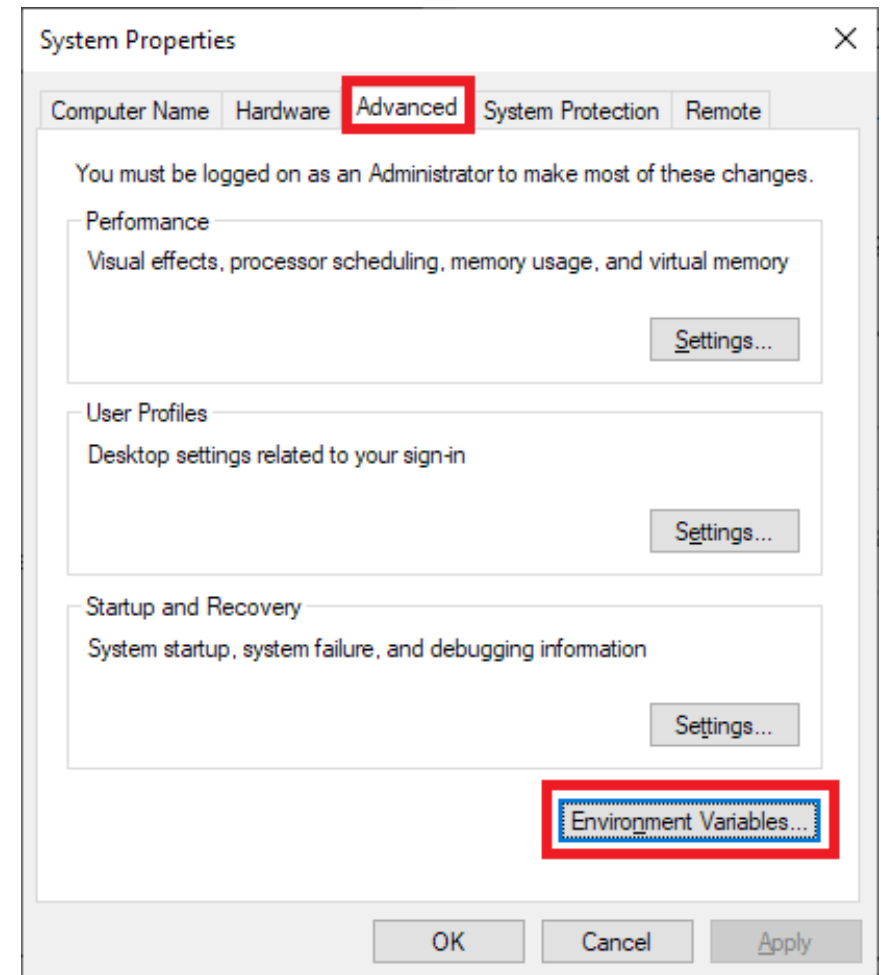
Editing PATH Environment Variable

On the "About" page (should be the default), click on "Advanced system settings" on the right



Editing PATH Environment Variable

Go to the "Advanced" tab (again, should be the default) and click on "Environment Variables..." at the bottom of the menu



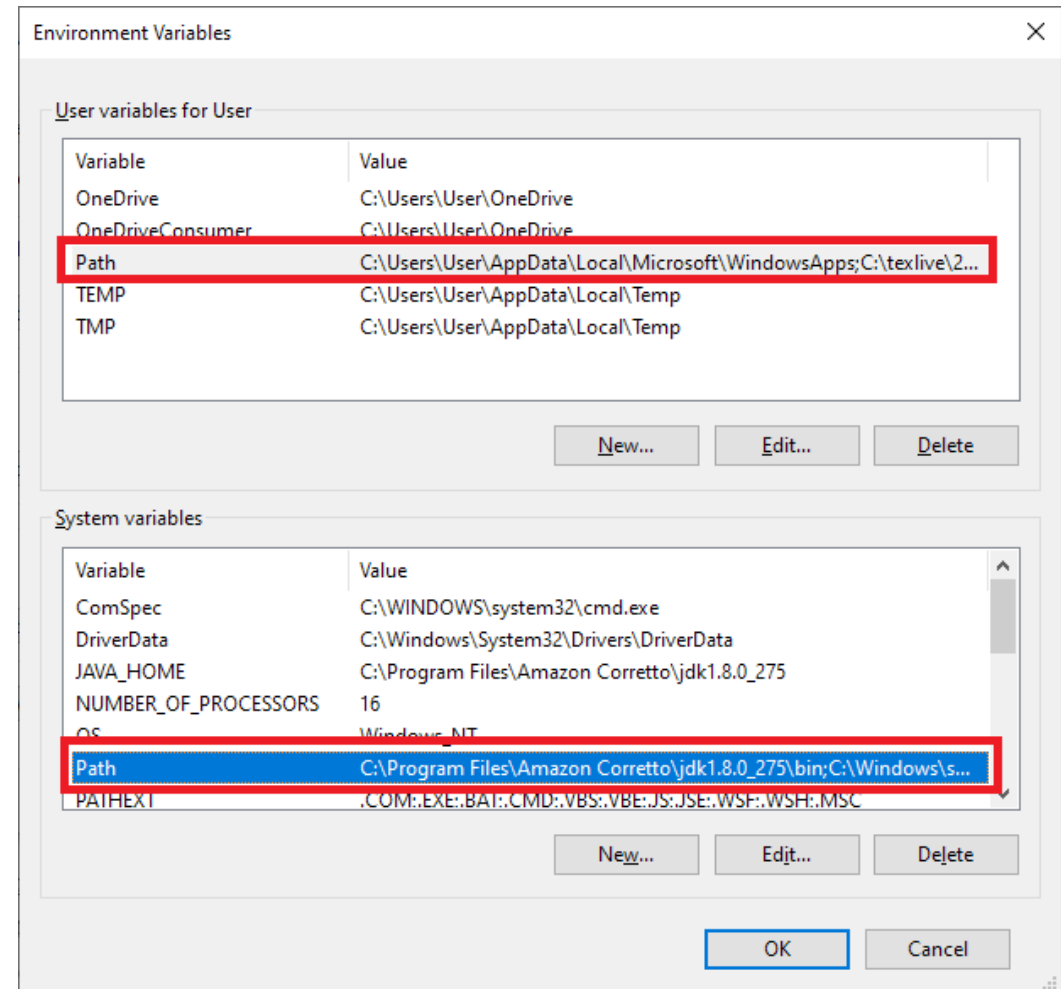
Editing PATH Environment Variable

From the following menu, you will find 2 different "Path" variables; editing either is fine.

Editing the "User" variables will only affect the current user.

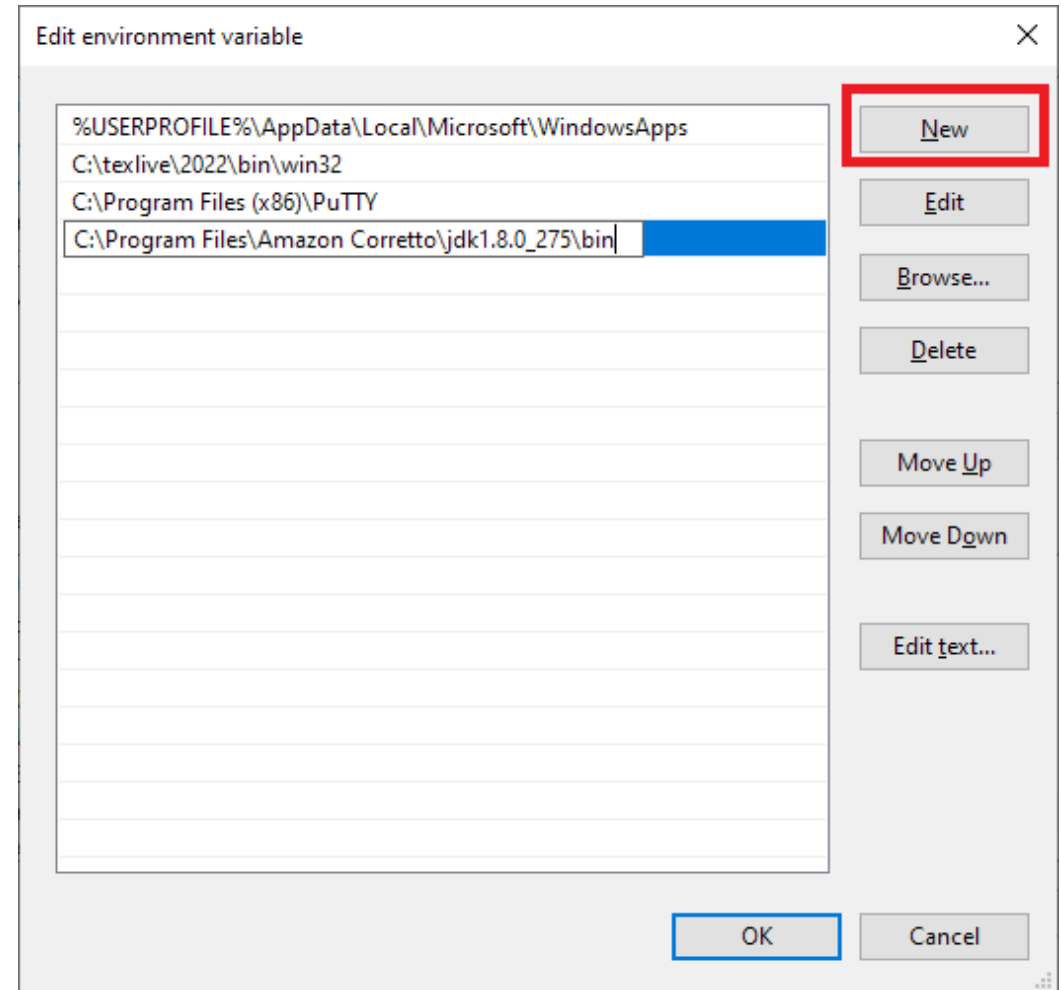
Editing the "System" variables will affect all users.

Editing can be done by double clicking that row.



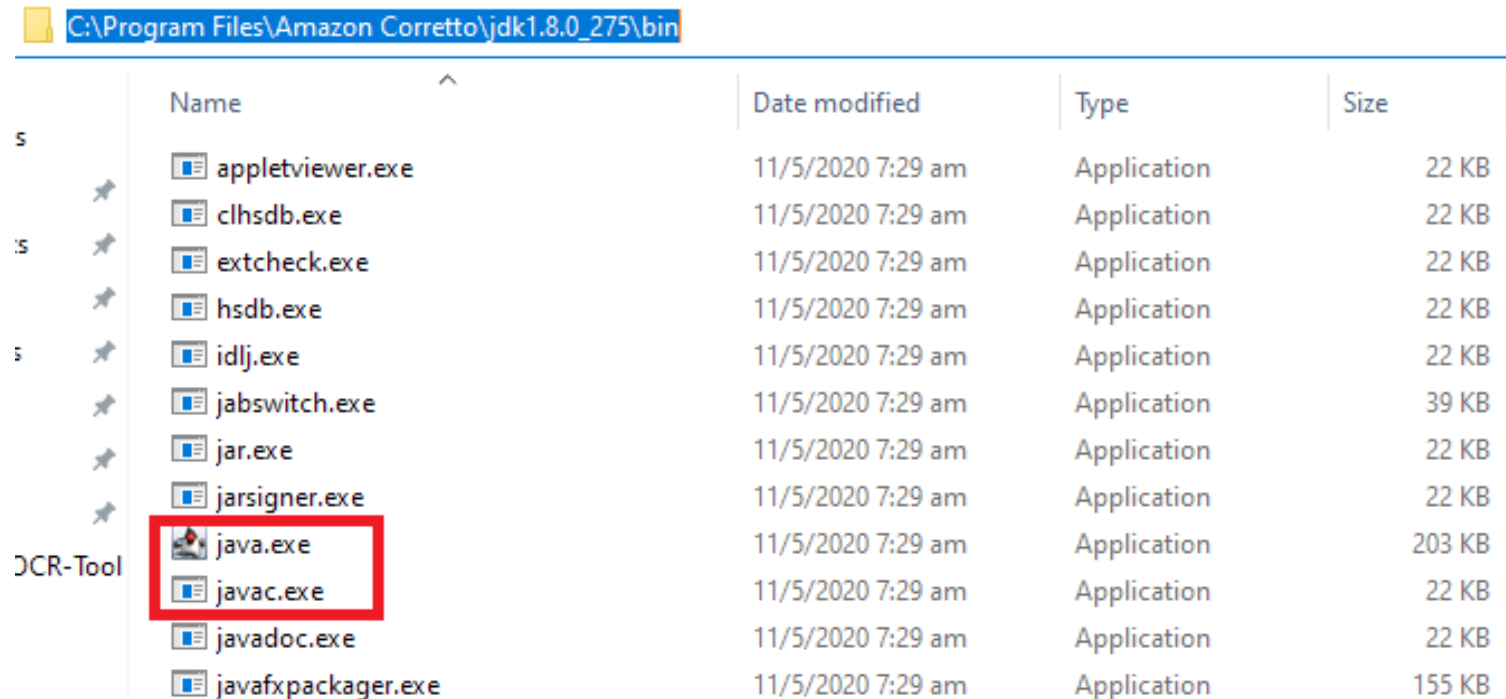
Editing PATH Environment Variable

In the window that follows, select "New" and enter the path of the folder containing "java.exe" and "javac.exe" for your device



Editing PATH Environment Variable

For this example, "java.exe" and "javac.exe" are found at "C:\Program Files\Amazon Corretto\jdk1.8.0_275\bin", but it may be different on your device



C:\Program Files\Amazon Corretto\jdk1.8.0_275\bin				
	Name	Date modified	Type	Size
	appletviewer.exe	11/5/2020 7:29 am	Application	22 KB
	clhsdb.exe	11/5/2020 7:29 am	Application	22 KB
	extcheck.exe	11/5/2020 7:29 am	Application	22 KB
	hsdb.exe	11/5/2020 7:29 am	Application	22 KB
	idlj.exe	11/5/2020 7:29 am	Application	22 KB
	jabswitch.exe	11/5/2020 7:29 am	Application	39 KB
	jar.exe	11/5/2020 7:29 am	Application	22 KB
	jarsigner.exe	11/5/2020 7:29 am	Application	22 KB
	java.exe	11/5/2020 7:29 am	Application	203 KB
	javadoc.exe	11/5/2020 7:29 am	Application	22 KB
	javafxpackager.exe	11/5/2020 7:29 am	Application	155 KB

Lab 1 – Command Line/Terminal Usage

- `cd ../../test`
 - “cd” is the command used to move between folders. “..” means to go up one folder
- `dir`
 - Used to list the contents of a folder. Use “/s” (lowercase L, lowercase S) on Unix-like systems or Powershell
- `javac Echo.java`
 - “javac” is the command use to compile Java programs. The full file name (inclusive of “.java”) is required, and seeing no output is a **good** thing.
 - Each class in the Java file is compiled to its own “.class” file

Lab 1 – Command Line/Terminal Usage

- `java Echo < Hello.txt`
- `java Echo < HelloDouble.txt`
 - “java” is the command used to run compiled Java programs. No filename extension (.java, .class) should be provided
 - “<” followed by the name of a file is used to provide input from a file (known as input redirection).
Instead of having to type out your input manually; simply save your input in a file and use “<” to avoid retyping input each time you test your program
- `java Echo > Bye.txt`
 - “>” followed by the name of a file is used to send output to the file (known as output redirection)
 - Can be used together with input redirection, e.g. *java Echo < Hello.txt > Bye.txt*

One-Day Assignment 0 – Pea Soup

- First one-day assignment is ungraded
- Found on the main Kattis course page at https://nus.kattis.com/courses/CS2040/CS2040_S1_AY2324.
- Writing pseudocode for this assignment is not necessary, but will be required from the next one-day assignment onwards

One-Day Assignment 0 – Pea Soup

- Given a list of restaurants, and their menus, determine the first restaurant appearing in the list that offers both “pea soup” and “pancakes”
- Should be the exact strings “pea soup” and “pancakes”; even if a menu item contains the substring “pea soup” or “pancakes”, ignore it (see Sample Input 2)

Assignment Guidelines

- **Important:** Include your **name** and **student number** in comments at the top of your code

Rehash of “Rules for CS2040 Assignments”:

- If you discuss the problem with any other student(s), include their name(s) as collaborators in a comment at the top of your code
- No usage of anyone else’s code (outside of code provided in lecture materials)
 - Directly using (eg. copy-paste) is not allowed
 - Using code as a reference is not allowed either

CS2040 Lab 2

Java Introduction (2)

One-Day Assignment 0 – Pea Soup

- Algorithm released on Canvas under Labs
- Future one-day assignments will have algorithms released on Canvas at 6pm on the day it is released, under Labs > Lab X
 - Intended to help students who may have difficulty coming up with the algorithm, so they can work on the assignment before it is due
 - Ideally, students should come up with the algorithm on their own (helps with written assessments, or the equivalent of it in e-learning form)
- Slides for the lab and other relevant files will also be found there
- Take-home assignments will not have the algorithm provided

One-Day Assignment 0 – Pea Soup

- Implementation wise, program should read in input correctly:
 - Due to the way `sc.nextInt()` and `sc.nextLine()` works, running an `sc.nextLine()` immediately after an `sc.nextInt()` may result in it reading in an empty string
 - Use an additional dummy `sc.nextLine()` to clear away the empty string first ie.
 - `int n = sc.nextInt();`
 - `sc.nextLine();`
 - `String input = sc.nextLine();`

123\n

hello bye\n

Lab 2 – Speed

- Some parts of Java may cause unintended slowness in your program
- Possible examples are use of slower (in terms of big O) API calls
- Other examples are of methods with the same big O time complexity, but has a higher “constant factor”
 - Eg. a $10n$ method call vs $2n$ method call, both are $O(n)$ but the $10n$ method would run slower
- Useful API and notes to address these issues are covered in the next few slides

Lab 2 – Buffered IO

- Scanner has convenient functions `nextInt()`, `nextDouble()` etc.
 - Is actually pretty slow
- Similarly, `System.out.print()/println()/printf()` may use up a lot of time if called repeatedly
- Other methods of handling IO functionality exist, which are faster but also a bit more complicated to use
- Some take-home assignments will require the use of buffered IO (using `Scanner/System.out` will result in exceeding the time limit)
 - All one-day assignments are guaranteed to be solvable just by using `Scanner/System.out`, though optimisations may be necessary in other parts of your program if this route is chosen

Lab 2 – BufferedReader

- Provides a much faster way to read in input
- Initialise using the following line (be sure to import java.io.* first)
 - `BufferedReader br = new BufferedReader(new InputStreamReader(System.in));`
- Provides very few methods to read in input; the most frequently used one would be `readLine()`, which behaves much like Scanner's `nextLine()`

Lab 2 – BufferedReader

Method name	Description	Time
<code>.readLine()</code>	Reads until it reaches the end of the line	$O(n)$

(yes, that's all, other methods exist but may not be as useful)

Lab 2 – BufferedReader

- `readLine()` will read in an entire line (similar to Scanner's `nextLine()`), so some methods may be useful for processing the result
 - Suppose the line we read in is : `String str = br.readLine();`
 - 1. Use `.split()` eg. `String[] strarr = str.split(" ");`
 - 2. Iterate over the array, using parse methods as necessary eg. `int num = Integer.parseInt(strarr[0]);`
 - Similar methods for other primitive data types exist eg. `Long.parseLong()`, `Double.parseDouble()`

1 2 3 4 5 (input)

"1 2 3 4 5" (`readLine`) -> ["1", "2", ...] (`split`) -> [1, 2, 3, ..] (`parse`)

Lab 2 – PrintWriter

- Provides a much faster way to write output
- Basically the same as System.out methods, but delays printing until a .flush() or .close() is called (to avoid repeated switching between printing and computation, thereby saving some time)
- Initialise using the following line (be sure to import java.io.* first)
 - `PrintWriter pw = new PrintWriter(new BufferedWriter(new OutputStreamWriter(System.out)));`

Lab 2 – PrintWriter

- Ideally, only call `.flush()` or `.close()` **once**, just before exiting your program
 - Not calling `.flush()` or `.close()` can result in lost output
 - Calling `.flush()` too many times can result in slow programs (eg. calling `.flush()` after each `.print()` statement makes it effectively the same as regular `System.out.print()` in terms of time taken)
 - You may want to do this during debugging, but remember to remove the extra `.flush()` calls later
 - Calling `.close()` before your program is ready to exit simply means that you cannot use that particular `PrintWriter` object to print anymore, and need to create another `PrintWriter` object (inefficient) to continue printing

Lab 2 – PrintWriter

Method name	Description	Time
<code>.print(String str)</code>	Prints <i>str</i>	$O(n)$
<code>.println(String str)</code>	Prints <i>str</i> , followed by a newline character (<code>'\n'</code>)	$O(n)$
<code>.printf(String str)</code>	Emulates the <code>printf</code> function of C	$O(n)$
<code>.flush()</code>	Flushes the buffer (ie. actually prints the contents of the writer to the screen)	$O(1)$
<code>.close()</code>	Calls <code>flush()</code> , then closes the writer. The writer cannot be used again	$O(1)$

Lab 2 – Kattio.java (suggestion is to use this one)

- Kattis provides its own version of a buffered IO, which uses the classes from earlier
- For input, it provides its own methods, covered in the next slide
- For output, it uses the same methods as `PrintWriter` (previous slide)
- Uploaded to Canvas > Files > Lab > Kattio.java
- Not part of the standard Java API, but you can use it by copy-pasting it into your own program, or submitting the file alongside your own program to Kattis

Lab 2 – Kattio.java

Method name	Description	Time
.getInt()	Reads the next token in the input as an integer	$O(n)$
.getLong()	Reads the next token in the input as a long	$O(n)$
.getDouble()	Reads the next token in the input as a double	$O(n)$
.getWord()	Reads the next token in the input as a String	$O(n)$

Lab 2 – Wrapper Classes

- Wrapper classes (eg. Integer, Double) act as a Java object version of primitive data types
 - Consider the example declarations below:
 - `int num1 = 1;`
 - `Integer num2 = 1;`
 - num1 contains the integer value 1
 - num2 contains a reference to a Java object, which contains the integer value 1
- Wrapper classes are immutable, requiring a new copy to be made each time a change is made

Lab 2 – Wrapper Classes

- Wrapper classes are convenient to use (no need to explicitly convert between a Java object and a primitive data type), but may have hidden performance issues, so use them only if necessary
 - Eg consider the statement $n = x + y$;
 - If n , x and y are all of type `int`, then the steps are:
 - Read the value of x , and the value of y , and sum them up. Put the resulting value into n
 - If n , x and y are all of type `Integer`, then the steps are:
 - Check the value of x , then read the object referenced by x . Access the `int` value stored in that object.
 - Then, check the value of y , then read the object referenced by y . Access the `int` value stored in that object
 - Sum the two values, then create a new object containing the result. Set the value of n to point to that object

Lab 2 – StringBuilder/StringBuffer

- As covered in lecture/tutorial, using the + operator, or the concat() method of a String will take $O(n + m)$ time, where n is the length of the first string, and m is the length of the second string
- To avoid this, we use the mutable string types
StringBuilder/StringBuffer
 - StringBuilder is intended for single-threaded applications (eg. programs in this module)
 - StringBuffer is intended for use in multi-threaded applications, and is slightly slower due to synchronisation
- The append operations would take $O(m)$ time, under the above definitions of n and m

Lab 2 – StringBuilder/StringBuffer

- Caution: the $O(m)$ time only applies to `.append()`
- Using `.insert()` (which allows adding to any position, not just the back of the `StringBuilder/StringBuffer`) takes $O((n-i) + m)$ time, where i is the index to be inserted to

Lab 2 – StringBuilder/StringBuffer

Method name	Description	Time
<code>.charAt(index i)</code>	Returns the character at index <i>i</i> (0-based)	$O(1)$
<code>.append(String other)[#]</code>	Adds <i>other</i> to the back of the stored string	$O(\text{length of } other)$
<code>.length()</code>	Returns the length of the stored string	$O(1)$
<code>.substring(int start, int end)</code>	Returns a new string, which contains the content of the original string from index <i>start</i> (inclusive) to index <i>end</i> (exclusive) (indices are 0-based)	$O(\text{length of resulting string})$
<code>.toString()</code>	Returns a copy of the stored string. Additional modifications to the StringBuilder/StringBuffer afterwards will not affect the copy (and vice versa)	$O(n)$

[#]Use the `.append()` method instead of `+` when trying to add on Strings to a StringBuilder/StringBuffer
The parameter can also be a `char[]`, or any primitive data type (int, long etc.)

StringBuilder/StringBuffer does not directly support `.compareTo()` (unlike String)

Lab 2 – StringBuilder/StringBuffer

- Eg. suppose we have an array of Strings, and wanted to add a line number to each of them, and join them into a single String:

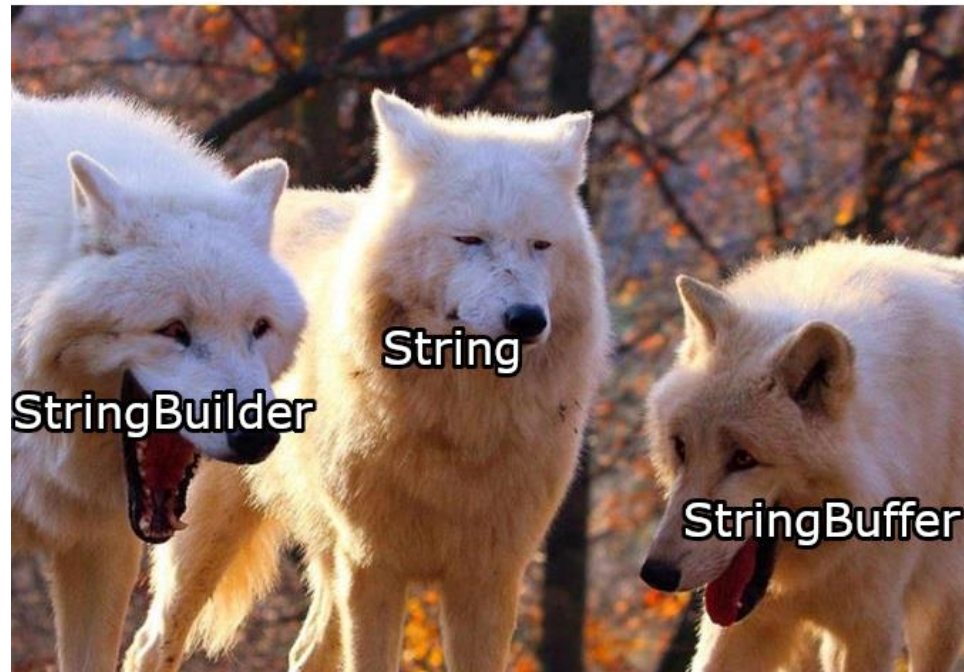
```
String str = ""; // empty string
for (int i = 0; i < arr.length; i++) {
    str = str + "Line " + i + ": " + arr[i] + "\n"; // slow execution time
}
```

- vs

```
StringBuilder sb = new StringBuilder(); // empty StringBuilder
for (int i = 0; i < arr.length; i++) {
    sb.append("Line ").append(i).append(": ").append(arr[i]).append("\n");
}
String str = sb.toString();
```


Lab 2 – StringBuilder/StringBuffer

Joining multiple strings,
one at a time in Java



One-Day Assignment 1 – T9 Spelling

- Reminder: All assignments should be submitted on nus.kattis.com, not open.kattis.com
- Remember to type out your pseudocode first before starting to code
- This assignment simulates old text systems for phones
- Each character has an associated series of button presses (eg. 'a' requires one press of the '2' button, 'b' requires 2 presses and 'c' requires 3, while 'd' is a single press of the '3' button) and so on
- Need to pause if typing two consecutive characters that both use the same button (eg. 'h' and 'l' both use the '4' button, so a pause is needed between the two)

One-Day Assignment 1 – T9 Spelling

- Hint: every character is actually represented as an integer from 0 to 255, known as its ASCII value
 - 'a' has an integer value of 97, 'b' has an integer value of 98 etc.
- Possible to “simulate” a dictionary (for Python/JavaScript users) of characters by creating an array of size 256, and using the character as the index
 - `String[] arr = new String[256];`
 - `String input = sc.nextLine(); // assume input is "cd"`
 - `arr['c'] = "222"; // or arr[99] = "222"`
 - `char letter = input.charAt(0);`
 - `System.out.println(arr[letter]); // prints 222`

One-Day Assignment 1 – T9 Spelling

- Also note: while the provided sample input for this question covers most of the special cases, you may want to think about cases that have not been covered, and are legal input, based on the definition in the question:
 - Cases covered:
 - Repeated letters from the same key: hello, hi
 - Repeated whitespaces: foo bar, where is a whitespace in the input
 - Cases not covered:
 - Strings starting/ending with whitespaces: ab
 - Strings consisting entirely of whitespaces:
 - Possible worst case scenario: a string consisting of 'z' 1000 times

Assignment Guidelines

- **Important:** Include your **name** and **student number** in comments at the top of your code

Rehash of “Rules for CS2040 Assignments”:

- If you discuss the problem with any other student(s), include their name(s) as collaborators in a comment at the top of your code
- No usage of anyone else’s code (outside of code provided in lecture materials)
 - Directly using (eg. copy-paste) is not allowed
 - Using code as a reference is not allowed either

CS2040 Lab 3

Sorting

One-Day Assignment 1 – T9 Spelling

- Reminder for reading input: when reading using Scanner's `nextLine()` method after a non-`nextLine()` method (eg. `next()`, `nextInt()`), using an additional `nextLine()` call may be necessary to avoid reading in an empty line
- In your program, do not create more than 1 object to read in input (ie. use only 1 Scanner, or only 1 BufferedReader instead), as both objects “buffer” input by reading in a large chunk of input and saving it in memory
 - See “ScannerInputDemo.java”

Lab 3 – Useful API

- Some sorting methods have already been implemented in Java API
- The sorting methods provided by Java are sufficient for general use.
As such, you may not need to code out your own sorting algorithms

Lab 3 – Useful API

- `Arrays.sort(arr)` will sort a primitive array (eg. `int[]`) *arr* using double-pivot quicksort
- However, if *arr* contains an object instead of primitive data types, it will use a sorting algorithm called TimSort (not examinable)
 - Is stable
 - Not in-place
 - Runs in worst case $O(n \log n)$ time
 - Runs in $O(n)$ time if array is almost sorted
- `Collections.sort(list)` will sort a List using TimSort
 - More on lists in the next lab

Lab 3 – Useful API

- For the sorting methods provided by the Java API to function, it needs to have a way to determine how one element relates to another
 - I.e. when comparing two elements, is the first element smaller than/greater than/equal to the second element?
- Primitive data types (int, double), their associated wrapper classes, and Strings already have this built in
 - Some other Java classes have this too, but you'll likely not use them in this module (eg. Date, Month, Year classes)
- For custom classes, you'll have to add this yourself

Lab 3 – Comparing.java Example

- The program “Comparing.java” is provided as an example of how to code out comparison methods (covered in the next few slides)
- Slides cover the more theoretical parts, which may be a little difficult to understand on their own

Lab 3 – Comparable Interface

- The Comparable interface is used by Java to determine that an object type has a built-in comparison method
 - Also referred to in the Java API documentation as *natural ordering*
- An object type that has a built-in comparison method should implement the Comparable interface
 - Doing so requires the interface's compareTo(T other) method to be implemented as well
 - T is a generic type

Lab 3 – Comparable Interface

- The `compareTo(T other)` method compares two objects: the object on which this method is called (ie. `this`), and the object passed in as a parameter
- The method should return an integer:
 - A negative integer if *this* < *other*
 - Zero if *this*, and *other*, are equivalent
 - A positive integer if *this* > *other*
- See array/list A1/B1 in Comparing.java for an example

Lab 3 – Comparator Interface

- The Comparator interface is another way to compare two objects
- Note that this is in part, a workaround for the Java programming language before Java 8; it did not support function passing then (ie. you can't pass in a function directly as a parameter)
 - Rather, you pass in an object, that contains a function
- The comparator should then be passed as a parameter into the sort() method

Lab 3 – Comparator Interface

- Passing in `Comparator.reverseOrder()` as a comparator will compare elements based on the reverse of the natural ordering
 - As such, the object stored in the array/list must already have implemented `Comparable`
- See array/list A2/B2 in `Comparing.java` for an example

Lab 3 – Comparator Interface

- You can also write a custom Comparator to compare two objects
- Need to implement the `compare(T first, T second)` method
 - The return value is similar to that in `compareTo`:
 - A negative integer if `first < second`
 - Zero if first and second are equivalent
 - A positive integer if `first > second`
- See array/list A3/B3 in `Comparing.java` for an example
 - Array/list A4/B4 is a shortcut of the above (declaring the comparator in the `sort()` method directly
 - Array/list A5/B5 is a shortcut of the above (lambda methods, may be a bit abstract for first-time use; recommended for advanced users)

Lab 3 – Sorting (Arrays)

Method name	Description	Time
<code>Arrays.sort(int[] arr)</code>	Sorts <i>arr</i> using double-pivot quicksort, if <i>arr</i> contains a primitive data type	$O(n \log n)$
<code>Arrays.sort(int[] arr, int start, int end)</code>	Sorts <i>arr</i> using double-pivot quicksort from <i>start</i> (inclusive) to <i>end</i> (exclusive), if <i>arr</i> contains a primitive data type	$O(n \log n)$, where n = size of range

No way to use a comparator for primitive data types

Lab 3 – Sorting (Arrays)

Method name	Description	Time
<code>Arrays.sort(YourClass[] arr)</code>	Sorts <i>arr</i> using Timsort, provided the array contains elements which implement the <i>Comparable</i> interface	$O(n \log n)$
<code>Arrays.sort(YourClass[] arr, int start, int end)</code>	Sorts <i>arr</i> using Timsort from <i>start</i> (inclusive) to <i>end</i> (exclusive), provided the array contains elements which implement the <i>Comparable</i> interface	$O(n \log n)$, where n = size of range
<code>Arrays.sort(YourClass[] arr, Comparator<YourClass> comp)</code>	Sorts <i>arr</i> using Timsort, using the provided comparator	$O(n \log n)$
<code>Arrays.sort(YourClass[] arr, int start, int end, Comparator<YourClass> comp)</code>	Sorts <i>arr</i> using Timsort from <i>start</i> (inclusive) to <i>end</i> (exclusive) using the provided comparator	$O(n \log n)$, where n = size of range

Lab 3 – Sorting (Collections)

Method name	Description	Time
<code>Collections.sort(List<YourClass> list)</code>	Sorts <i>list</i> using Timsort, provided the list contains elements which implement the <i>Comparable</i> interface	$O(n \log n)$
<code>Collections.sort(List<YourClass> list, Comparator<YourClass> comp)</code>	Sorts <i>list</i> using Timsort using the provided comparator	$O(n \log n)$

No way to sort only within a given range for lists using API

Take-Home Assignment 1a – Best Relay Team

- Given a list of runners, and their times as the first runner/subsequent runners, find the team arrangement that would result in the shortest time taken
- Trying all possible permutations of 4 runners would take too long ($500C4$ (choose 4 different runners) * $4C1$ (choose 1 runner to take the first 100m)) = 10 billion+, when $n = 500$
- Can we try to find all permutations of a smaller subset of runners instead?
 - If so, is there an easy way to determine which runners we should consider?

Take-Home Assignment 1b – Card Trading

- Given T card types, their buy/sell prices, and the N initial cards Anthony has in his deck, determine the maximum amount of money that can be earned while keeping at least 2 or more cards for K different card types
- Only one of the following can be done for each card type:
 - Buy up to 2 cards of that type
 - Sell all (owned) cards of that type

Take-Home Assignment 1b – Card Trading

- The "int" data type may be insufficient for this question (its range is up to 2.1 billion); consider using "long" instead
 - Since buy/sell prices can be up to 1 billion, with 100,000 different card types, the maximum answer could be around 200 trillion
- Question asks for a deck with exactly K types of cards which Anthony owns more than 2 of
 - Does Anthony need to have more than 2 cards of any given type?
 - No, having more than 2 cards is unnecessary
 - Should Anthony end up with any card types which he has only one card for?
 - No, as it does not contribute to a combo, this should not be part of the final deck

Take-Home Assignment 1b – Card Trading

- Anthony can start off owning pairs of multiple card types already
 - Should Anthony keep *all* of his starting pairs of cards to form a complete deck?
 - No, it may be possible to sell off some cards which have a high selling price, in order to buy even more cards with a cheaper buying price
 - Otherwise, should Anthony sell off *all* of his starting cards?
 - No, some card types have a low selling price, so it may be better to keep them as a combo instead
 - It might help to consider how much it "costs" to keep a card type as a combo, instead of selling it

One-Day Assignment 2 – Sort of Sorting

- Given a list of names (strings)
- Sort them according to the first two letters of their names
 - Guaranteed that name has at least two letters
 - Note: the default comparison method for strings uses the entire string
- Some form of stable sort required

One-Day Assignment 2 – Sort of Sorting

- Again, consider cases which are covered by the sample input, and corner cases which may not be covered:
- Covered:
 - Names with the same first 2 letters:
 - Poincare
 - Pochhammer
- Not covered:
 - Comparing an uppercase letter with a lowercase letter
 - Zoe
 - amy

CS2040 Lab 4

List ADT

One-Day Assignment 2 – Sort of Sorting

- Need to use stable sort
- Thankfully, Java's Arrays.sort() is stable by default
- Just write a custom comparator to compare by first 2 characters, instead of the entire string

```
// the overridden compare method
public int compare(String a, String b) {
    return a.substr(0, 2).compareTo(b.substr(0, 2));
}
```

Take-Home Assignment 1a – Best Relay Team

- Use a custom class to represent a Runner
 - Contains the 1st leg timing, subsequent leg timings and name
- Use 2 arrays to store the Runners
 - Sort one by 1st leg timing
 - Sort the other by subsequent leg timings
- Pick the first runner A from the 1st leg timing array
- Pick the first 3 runners from the subsequent leg timing array, that are not the same as runner A(since it is possible for the same runner to appear in the front of both arrays)
- Repeat the above for the second/third/fourth runner from the 1st leg timing array
- Output the team that gives the best timing among the 4 possible options

Take-Home Assignment 1b – Card Trading

- Start by assuming all cards Anthony owns are sold
 - This would result in the highest possible profit, if no other restrictions exist
- Then, determine the cost for keeping a certain card type as a combo
 - For a card type which Anthony has none of, he needs to buy 2 cards ($2 * \text{buy}$)
 - For a card type which Anthony has one of, he cannot sell that card, and has to buy one more ($1 * \text{sell}$, and $1 * \text{buy}$)
 - For a card type which Anthony has two of, he cannot sell both cards ($2 * \text{sell}$)
- Sort the cost for all card types, and pick the K smallest elements as the card types to keep. Subtract the sum of these elements from the profit (derived earlier) to get the final answer

Lab 4 – Generics

- Most Java API usage from this point on requires the use of generics
- Essentially a way to define the data type being used, much like arrays
- Eg. `String[]` versus `ArrayList<String>`
- Can also be used for your own custom classes
- Eg. `Student[]` versus `ArrayList<Student>`
- Note that primitive data types (eg. `int`, `double`) cannot be used as generics; you will need to use their relevant Java classes instead
 - Eg. `ArrayList<Integer>`, `ArrayList<Double>`

Lab 4 – Lists

- Two kinds of lists tend to be more frequently used in Java: ArrayList and LinkedList
- Main differences:
 - Accessing elements within the list
 - ArrayList offers $O(1)$ time to access any element in the list
 - LinkedList offers $O(1)$ time to access elements at the front and back of the list only
 - Inserting/deleting elements from the list
 - ArrayList offers $O(1)$ time to insert and delete from the back of the list only
 - LinkedList offers $O(1)$ time to insert and delete from the front or the back of the list
- Inserting/deleting from other positions (apart from the ones mentioned above) runs in about $O(n)$ time for both ArrayList and LinkedList
- For LinkedList accessing any position other than front and back result in $O(n)$ on average

Lab 4 – Lists

- For arrays, a Java class called Vector exists
- This is similar to the ArrayList class, but is “synchronised”
 - I.e. used for multi-threading, by ensuring that only one thread can modify the Vector at a given time
 - However, performing this check makes it slightly slower as a result
- For this module, just the use of ArrayList is sufficient, as we do not cover multi-threading

Lab 4 – ArrayList

Method name	Description	Time
.add(YourClass element)	Adds <i>element</i> to the end of the list	O(1)
.add(int index, YourClass element)	Adds <i>element</i> to the position at <i>index</i>	O(size() - index)
.clear()	Empties the list	O(n)
.contains(Object o)	Checks if <i>o</i> is in the list, based off the object's equals() method	O(n)
.get(int index)	Gets the element at <i>index</i>	O(1)
.remove(int index)	Removes the element at <i>index</i>	O(size() - index)
.size()	Returns the number of elements in the list	O(1)

O(size() - index) is due to the need to shift all elements to the right of *index* when inserting/deleting

Lab 4 – LinkedList

Method name	Description	Time
.add(YourClass element)	Adds <i>element</i> to the end of the list	O(1)
.add(int index, YourClass element)	Adds <i>element</i> to the position at <i>index</i>	O(size() - index) or O(index), whichever is smaller
.clear()	Empties the list	O(n)
.contains(Object o)	Checks if <i>o</i> is in the list, based off the object's equals() method	O(n)
.get(int index)	Gets the element at <i>index</i>	O(size() - index) or O(index), whichever is smaller
.remove(int index)	Removes the element at <i>index</i>	O(size() - index) or O(index), whichever is smaller
.size()	Returns the number of elements in the list	O(1)

Lab 4 – Stacks/Queues

- Special kinds of lists
 - Stacks only allow inserting, accessing and deleting from the top of the stack in $O(1)$ time (first-in last-out)
 - Queues only allow accessing and deleting from the front of queue, and inserting from the back of the queue in $O(1)$ time (first-in first-out)

Lab 4 – Stacks/Queues

- Queue is an interface, so you cannot initialize it using `new Queue()`
 - Instead, initialise it using a LinkedList: `Queue<YourClass> queue = new LinkedList<YourClass>();`
 - Or as an ArrayDeque (check the documentation)
- Stack is not an interface, so you can use `new Stack()` directly
 - `Stack<YourClass> stack = new Stack<YourClass>();`
- Note that the Stack class in Java extends the Vector class, so the use of methods such as `get(index)`, and `add(index, element)` is possible; however, these methods should not be used in a pure stack implementation

Lab 4 – Stack

Method name	Description	Time
.empty()	Checks if the stack is empty	O(1)
.peek()	Retrieves the element at the top of the stack	O(1)
.push(YourClass element)	Adds <i>element</i> to the top of the stack	O(1)
.pop()	Removes and retrieves the element at the top of the stack	O(1)

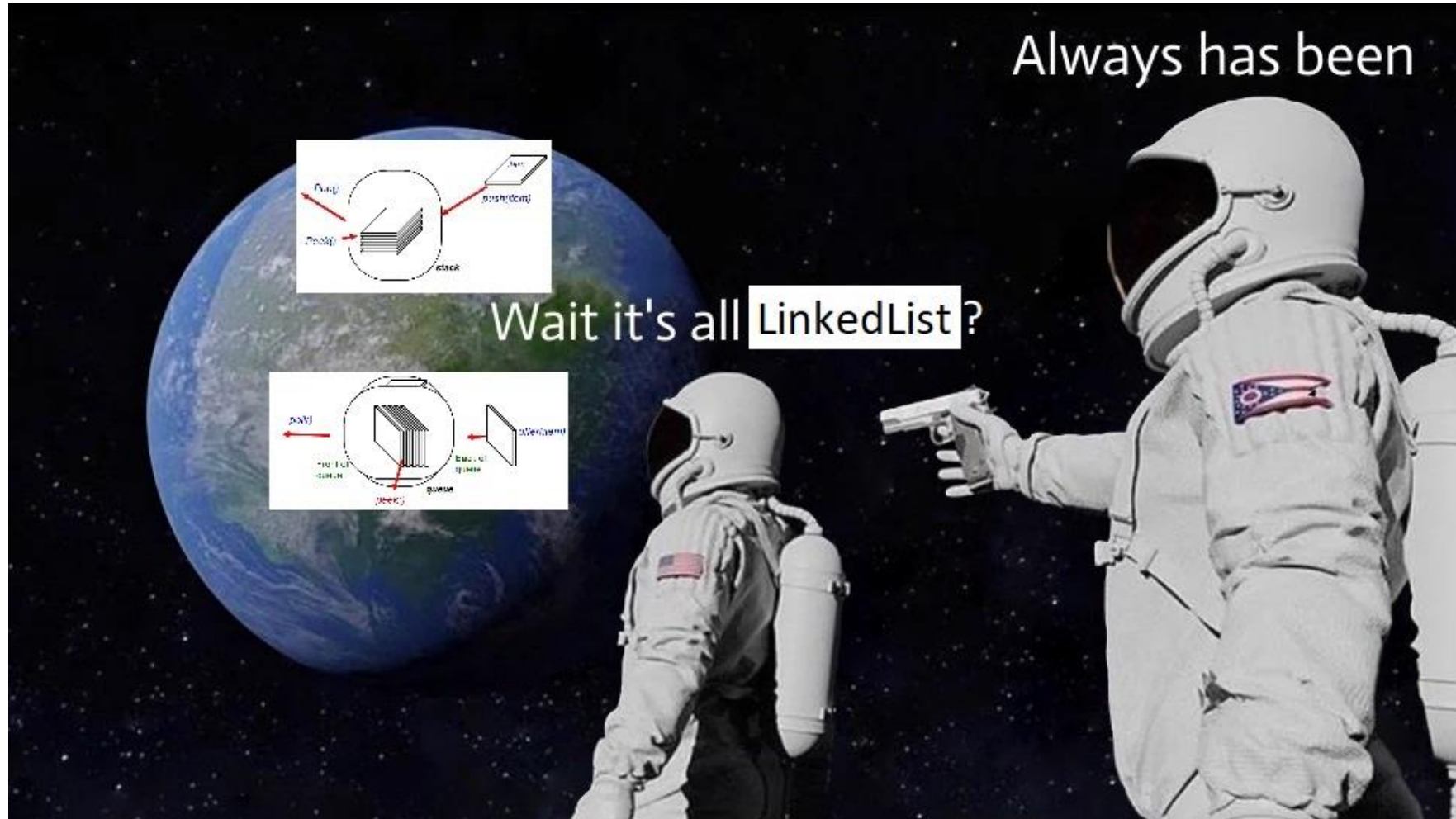
Lab 4 – Queue (using LinkedList)

Method name	Description	Time
<code>.isEmpty()</code>	Checks if the queue is empty	$O(1)$
<code>.offer(YourClass element)</code>	Adds <i>element</i> to the end of the queue	$O(1)$
<code>.peek()</code>	Retrieves the element at the head of the queue	$O(1)$
<code>.poll()</code>	Removes and retrieves the element at the head of the queue	$O(1)$

Lab 4 – LinkedList (again)

- Note that LinkedList supports operations from both Stack (excluding the .empty() method) and Queue
 - However, LinkedList does not extend Stack, so the following is **not** possible:
 - `Stack<YourClass> stack = new LinkedList<YourClass>();`
- As such, it's entirely possible to declare LinkedLists anytime you need a stack or a queue:
 - The top of the stack is the head of the LinkedList
 - The front of the queue is the head of the LinkedList, while the back is the tail of the LinkedList
 - The .peek() method is consistent with both definitions (returns the element at the head of the list)

Lab 4 – LinkedList (again)



Take-Home Assignment 2a – Join Strings

- Given a list of strings, concatenate them together, and output the final string
- Main emphasis of the question is on concatenating strings together
 - Concatenating strings (via `str.concat()` or the `+` operator) takes $O(k)$ time, where k is the length of the resulting string (see Tutorial 1, problem 2e)
 - Using `StringBuilder/StringBuffer`'s `.append()` is slightly better (takes $O(m)$ time, where m is the length of the string that is added on), but still too slow
 - Therefore, simply using the default string methods will not be fast enough
 - Need to find a way to do so in $O(1)$ time when handling queries

Take-Home Assignment 2b – Teque

- Create a custom data structure to support the `push_back`, `push_front`, `push_middle`, and `get` operations
- Need to ensure all operations run in $O(1)$ time
- This problem will also require the use of buffered IO methods (covered in Lab 2 slides), due to the large input/output sizes (up to 1M lines!)

One-Day Assignment 3 – Coconut Splat

- Simulate a counting-out game
- A player's hand has 4 possible states:
 - 1. Both hands folded together (initial state)
 - 2. Fist
 - 3. Palm down
 - 4. Behind back
- A player's hand moves from state n to state $(n + 1)$ if that hand is touched last
- A player is out of the game if both hands are behind their back
- Find out which player will be the last player standing

One-Day Assignment 3 – Coconut Splat

- State 1 (folded), initial state:
 - Counts as a set of hands (ie. only one syllable) in this form instead of two individual hands
- State 1 (folded) -> State 2 (fist)
 - Split folded hands into 2 fists; counting begins with the first fist on the next round (so it goes to the player's first fist, followed by the player's second fist, followed by the hand of the next player)
- State 2 (fist) -> State 3 (palm down)
 - Change fist into palm down; counting begins with the next hand after this (which could be the player's other hand, or the next player's hand)
- State 3 (palm down) -> State 4 (behind back)
 - Hand is moved behind back; this hand will no longer be counted in subsequent rounds. Counting begins with the next hand after this (which could be the player's other hand, or the next player's hand)

One-Day Assignment 3 – Coconut Splat

- The following 2 slides contain an example of the game simulated for 3 players, with 3 syllables in the rhyme
- The last person standing in this example should be player 2
- In the example, each round starts from the underlined hand/fist/palm

Hand is touched		Hand is touched (last)		Hand changes state		Hand is out of play	
Move	Player 1		Player 2		Player 3		
Initial State	<u>Folded Hands</u>		Folded Hands		Folded Hands		
Round 1	Folded Hands		Folded Hands		Folded Hands		
After R1	Folded Hands		Folded Hands		<u>Fist</u>	Fist	
Round 2	Folded Hands		Folded Hands		Fist	Fist	
After R2	<u>Fist</u>	Fist	Folded Hands		Fist	Fist	
Round 3	Fist	Fist	Folded Hands		Fist	Fist	
After R3	Fist	Fist	<u>Fist</u>	Fist	Fist	Fist	
Round 4	Fist	Fist	Fist	Fist	Fist	Fist	
After R4	Fist	Fist	Fist	Fist	Palm Down	<u>Fist</u>	
Round 5	Fist	Fist	Fist	Fist	Palm Down	Fist	
After R5	Fist	Palm Down	<u>Fist</u>	Fist	Palm Down	Fist	
Round 6	Fist	Palm Down	Fist	Fist	Palm Down	Fist	
After R6	Fist	Palm Down	Fist	Fist	Behind Back	<u>Fist</u>	
Round 7	Fist	Palm Down	Fist	Fist	Behind Back	Fist	
After R7	Fist	Behind Back	<u>Fist</u>	Fist	Behind Back	Fist	

Hand is touched		Hand is touched (last)		Hand changes state		Hand is out of play	
Move	Player 1		Player 2		Player 3		
After R7	Fist	Behind Back	<u>Fist</u>	Fist	Behind Back	Fist	
Round 8	Fist	Behind Back	Fist	Fist	Behind Back	Fist	
After R8	<u>Fist</u>	Behind Back	Fist	Fist	Behind Back	Palm Down	
Round 9	Fist	Behind Back	Fist	Fist	Behind Back	Palm Down	
After R9	Fist	Behind Back	Fist	Palm Down	Behind Back	<u>Palm Down</u>	
Round 10	Fist	Behind Back	Fist	Palm Down	Behind Back	Palm Down	
After R10	Fist	Behind Back	Palm Down	<u>Palm Down</u>	Behind Back	Palm Down	
Round 11	Fist	Behind Back	Palm Down	Palm Down	Behind Back	Palm Down	
After R11	Palm Down	Behind Back	<u>Palm Down</u>	Palm Down	Behind Back	Palm Down	
Round 12	Palm Down	Behind Back	Palm Down	Palm Down	Behind Back	Palm Down	
After R12	<u>Palm Down</u>	Behind Back	Palm Down	Palm Down	Behind Back	Behind Back	
Round 13	Palm Down	Behind Back	Palm Down	Palm Down	Behind Back	Behind Back	
After R13	<u>Palm Down</u>	Behind Back	Palm Down	Behind Back	Behind Back	Behind Back	
Round 14	Palm Down	Behind Back	Palm Down	Behind Back	Behind Back	Behind Back	
After R14	Behind Back	Behind Back	<u>Palm Down</u>	Behind Back	Behind Back	Behind Back	

X_X

^_^

Deque (extra)

- A subinterface of Queue
- Supports insertion and removal at both ends, hence a “double-ended queue”
- Initialised with either LinkedList or ArrayDeque
- Can also use it as a Stack or Queue
 - In both cases, the first element to be removed is the first (ie. leftmost) element in the data structure

Deque (extra)

Method name	Description	Time
.isEmpty()	Checks if the deque is empty	O(1)
.peek()	Retrieves, but does not remove, the first element of this deque.	O(1)
.offer(YourClass element)	Inserts the specified element into the queue represented by this deque.	O(1)
.poll()	Retrieves and removes the head of the queue represented by this deque.	O(1)
.push(YourClass element)	Pushes an element onto the stack represented by this deque. (i.e. head)	O(1)
.pop()	Pops an element from the stack represented by this deque.	O(1)

CS2040 Lab 5

Hashing

Take-Home Assignment 2a – Join Strings

- Use custom linked list implementation (TailedLinkedList.java suffices)
- Use `ArrayList<TailedLinkedList>` (or a regular `TailedLinkedList[]` array)
- When concatenating string `b` to `a` (ie. string is `a + b`), set tail node of `a` to point to head node of `b`
- Update tail node pointer of `a` to tail node of `b`
- No need to set `b` to null in the array after concatenation, as `b` is never referenced again (but you can do so if you want)
- Use `StringBuffer` or `StringBuilder` when constructing the final answer, or just print directly without creating a string

Take-Home Assignment 2b – Teque

- All operations need to run in $O(1)$ time
- For access to an element in $O(1)$ time (get() operation), this suggests the use of an array
- Pushing to the back can easily be done in $O(1)$ time too
- For adding to the front, instead of fixing index 0 as the front, allow for the front index to change
- This can be accomplished by using a circular array, with head and tail indices, as well as size attribute

Take-Home Assignment 2b – Teque

- How to insert to middle?
- Split into two circular arrays, one contains the front half (array 1), while the other contains the back half of the ADT (array 2)
- **Balance out** the number of elements between the two arrays after every push operation if necessary, so the middle is easily accessible
 - Should never need to move more than 1 element between the two arrays, so this is still $O(1)$
- For a `get(n)` operation using 2 arrays, access the n th element in array 1, or the $(n - (\text{size of array 1}))$ th element in array 2 if n exceeds the size of array 1

Lab 5 – Hashing

- Two different forms of hash tables are provided by Java
- **HashMaps** involve a key-value pair
 - The key is the object which is hashed, and then put into the HashMap depending on its resulting hash code
 - The value is a separate object that is associated with that key, and its contents are not considered when computing the hash code of the key
 - The entire key-value pair is called an entry
- **HashSets** simply involve one object, that acts as both the key and the value
 - Similar to the mathematical idea of set

Lab 5 – Hashing

- Note: there is also a Hashtable class which behaves in the same manner as a HashMap, but is generally slower than HashMap for the purposes of CS2040S
 - The reason is similar to ArrayList vs Vector ie. synchronisation

Lab 5 – HashSet

Method name	Description	Time
<code>.add(YourClass element)</code>	Adds <i>element</i> to the HashSet	O(1) average
<code>.clear()</code>	Clears the HashSet	O(n)
<code>.contains(Object o)</code>	Checks if <i>o</i> is in the HashSet, based off the object's <code>equals()</code> method	O(1) average
<code>.isEmpty()</code>	Checks if the HashSet is empty	O(1)
<code>.remove(Object o)</code>	Removes <i>o</i> if it is in the HashSet, based off the object's <code>equals()</code> method	O(1) average
<code>.size()</code>	Returns the number of elements in the HashSet	O(1)

Lab 5 – HashMap

- HashMaps are the first class encountered in this module that require the use of more than one generic
- To declare a HashMap with an Integer as the key, and a String as the value:
 - `HashMap<Integer, String> map = new HashMap<Integer, String>();`
- The two generics used need not be different eg. to use an integer as the key, and another integer as the value:
 - `HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();`

Lab 5 – HashMap

- HashMaps are generally useful in answering queries where the key is known, but the value is unknown
 - Eg. if we want to know which stall at a food court sells a certain food item, we could store a key-value pair of all foods, where the key is the food item's name, and the value is the stall name.
 - If we have a craving for a certain food item later, we use the food item's name to look up the HashMap, and check the value (stall name) associated with that key
- Effectively, using the key as a “reference” for the value

Lab 5 – HashMap

Method name	Description	Time
<code>.put(YourClass key, YourClass value)</code>	Adds <i>key</i> to the HashMap with the value <i>value</i>	O(1) average
<code>.clear()</code>	Clears the HashMap	O(n)
<code>.containsKey(Object o)</code>	Checks if key <i>o</i> is in the HashMap, based off the object's <code>equals()</code> method	O(1) average
<code>.containsValue(Object o)</code>	Checks if value <i>o</i> is in the HashMap, based off the object's <code>equals()</code> method	O(n)
<code>.get(Object o)</code>	Gets the value corresponding to the key <i>o</i>	O(1) average
<code>.isEmpty()</code>	Checks if the HashMap is empty	O(1)
<code>.remove(Object o)</code>	Removes the entry with key <i>o</i> if it is in the HashMap, based off the object's <code>equals()</code> method	O(1) average
<code>.size()</code>	Returns the number of elements in the HashMap	O(1)

Lab 5 – HashMap (cont.)

Method name	Description	Time
.entrySet()	Returns a set of all entries in the HashMap	O(1)
.keySet()	Returns a set of all keys in the HashMap	O(1)
.values()	Returns a collection of all values in the HashMap	O(1)

Unlike HashSet, it is not possible to iterate through a HashMap (eg. enhanced for-loop) directly. You will need to use the above methods to access an iterable form of the data stored within

One-Day Assignment 4 – Conformity

- Given the course combinations of students, determine how many students are taking one of the most popular combination of courses
 - There can be multiple combinations of courses which are tied for the most popular combination eg:
 - (100, 101, 102, 103, 104) – 4 students
 - (101, 102, 103, 104, 105) – 2 students
 - (102, 103, 104, 105, 106) – 4 students
 - Your answer should be $4 + 4 = 8$ (the students taking the first and third combinations)
- Note that course combinations can be given in different orders, but should be considered as the same course combination
 - Eg (100, 101, 102, 103, 488) is the same combination as (103, 102, 101, 488, 100) as given in Sample Input 1