CS2040S Lab 2
Java Introduction (2)

One-Day Assignment 0 – Pea Soup

- Algorithm released on LumiNUS under Labs
- Future one-day assignments will have algorithms released on LumiNUS 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();`

    - `1`
    - `hello`

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
.readLine()	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()

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)));`
- Always call .flush() or .close() on the PrintWriter before exiting your program, or some output may not be printed

Lab 2 – PrintWriter

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

Lab 2 – Kattio.java

- 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)
- Found at https://nus.kattis.com/help/java - search for “Kattio.java”
  - Direct link to the file not provided, as the link may change if Kattis updates the file
- 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

Method name	Description	Time
.charAt(index i)	Returns the character at index *i* (0-based)	O(1)
.append(String other)#	Adds *other* to the back of the stored string	O(length of *other*)
.length()	Returns the length of the stored string	O(1)
.substring(int start, int end)	Returns a new string, which contains the content of the original string from index *start* (inclusive) to index *end* (exclusive) (indices are 0-based)	O(length of resulting string)
.toString()	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

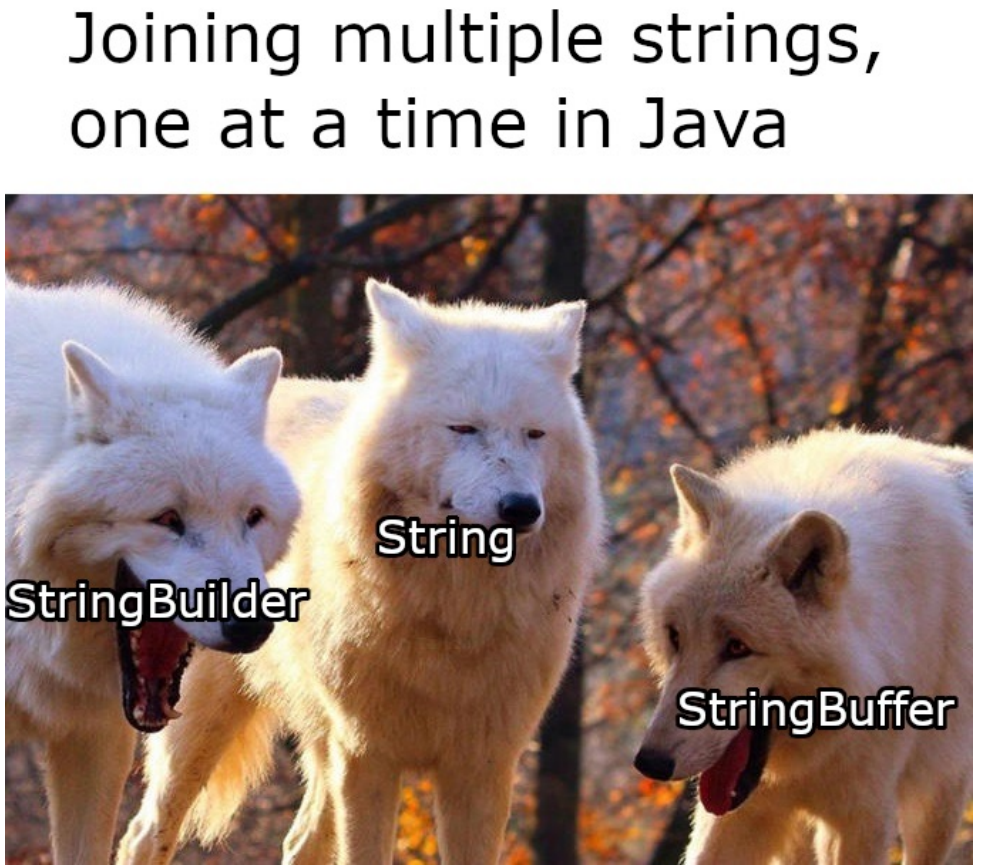
- Eg. suppose we have an array of Strings, and wanted to add a line number to each of them, and join them into a single String:

```
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



One-Day Assignment 1 – T9 Spelling

- Reminder: All assignments should be submitted on nus.kattis.com, not open.kattis.com
- 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 ‘i’ 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”;`
  - `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