CS2040S Lab 2

Java Introduction II

Pea Soup

Clear remainder of line, before switching to line-base mode.

Track if we have seen “pea soup” and “pancakes” yet.

If item matches what we want, mark it as seen.

If we have seen both in this menu, print this restaurant, **and then <u>immediately finish</u>**.

If we haven’t found anything, print “Anywhere…”

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

for(int i=0; i<n; i++) {
    int k = s.nextInt(); s.nextLine();

    String name = s.nextLine();
    boolean seenPeas = false,
        seenPancakes = false;

    for(int j=0; j<k; j++) {
        String item = s.nextLine();
        if("pea soup".equals(item)) {
            seenPeas = true;
        } else if("pancakes".equals(item)) {
            seenPancakes = true;
        }
    }

    if(seenPeas & seenPancakes) {
        System.out.println(name);
        return;
    }
}
System.out.println("Anywhere is fine I guess");
```

Common Mistakes: Input Format

- Ensure when you test
  - What you key in, matches the input format **exactly.**
  - Your output **exactly** matches the expected output.
    - A missing punctuation mark may be tiny, but makes all the difference.
    - An extra space or newline usually is tolerated

Common Mistakes: Input Format

- **Always** read the specification of the input format
- The problem states:
  - “Strings consist only of lower case letters ‘a’-‘z’ and **<u>spaces</u>**, and they always start and end with a letter.”
- Cannot use `next()` to read names.

Input Format

Names may contain spaces, according to the problem statement.

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

for(int i=0; i<n; i++) {
     int k = s.nextInt(); s.nextLine();

     String name = s.next();
     boolean seenPeas = false,
          seenPancakes = false;

     for(int j=0; j<k; j++) {
          String item = s.next();
          if("pea soup".equals(item)) {
               seenPeas = true;
          } else if("pancakes".equals(item)) {
               seenPancakes = true;
          }
     }

     if(seenPeas & seenPancakes) {
          System.out.println(name);
          return;
     }
}
System.out.println("Anywhere is fine I guess");
```

Common Mistakes: Scanner

- `nextInt()/nextDouble()/next()`
  - Token/word-based
  - Up to next whitespace
- `nextLine()`
  - Line-based
  - Up to next newline
- Token/word-based methods may leave leftover bits of the current line

Scanner

After reading a word/token, there may be remainder of line left over.

This `nextLine()` call will read the left-over, not the value you want.

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

for(int i=0; i<n; i++) {
    int k = s.nextInt(); // s.nextLine();

    String name = s.nextLine();
    boolean seenPeas = false,
        seenPancakes = false;

    for(int j=0; j<k; j++) {
        String item = s.nextLine();
        if("pea soup".equals(item)) {
            seenPeas = true;
        } else if("pancakes".equals(item)) {
            seenPancakes = true;
        }
    }

    if(seenPeas & seenPancakes) {
        System.out.println(name);
        return;
    }
}
System.out.println("Anywhere is fine I guess");
```

Common Mistakes: String Equality

- String/Object equality in Java
  - `a == b` tests **reference** equality
    - If a and b point to the **same object in memory**
    - `"" == ("abc".substring(0,0))` (may) return false!
      - Even if both are “empty string” values
  - `a.equals(b)` tests **value** equality
    - If a and b have **equal values**
    - Or `Objects.equals(a, b)` if a is possibly null

String Equality

== compares exact String objects!
May not return true, even if same value.

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

for(int i=0; i<n; i++) {
     int k = s.nextInt(); s.nextLine();

     String name = s.nextLine();
     boolean seenPeas = false,
          seenPancakes = false;

     for(int j=0; j<k; j++) {
          String item = s.nextLine();
          if("pea soup" == item) {
               seenPeas = true;
          } else if("pancakes" == item) {
               seenPancakes = true;
          }
     }

     if(seenPeas & seenPancakes) {
          System.out.println(name);
          return;
     }
}
System.out.println("Anywhere is fine I guess");
```

Control Flow

If “pea soup”/“pancakes” appears in previous menus, it may contaminate into future menus!

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

boolean seenPeas = false,
        seenPancakes = false;
for(int i=0; i<n; i++) {
        int k = s.nextInt(); s.nextLine();

        String name = s.nextLine();

        for(int j=0; j<k; j++) {
                String item = s.nextLine();
                if("pea soup".equals(item)) {
                        seenPeas = true;
                } else if("pancakes".equals(item)) {
                        seenPancakes = true;
                }
        }

        if(seenPeas & seenPancakes) {
                System.out.println(name);
                return;
        }
}
System.out.println("Anywhere is fine I guess");
```

Control Flow

```
Scanner s = new Scanner(System.in);
int n = s.nextInt();

for(int i=0; i<n; i++) {
     int k = s.nextInt(); s.nextLine();

     String name = s.nextLine();
     boolean seenPeas = false,
          seenPancakes = false;

     for(int j=0; j<k; j++) {
          String item = s.nextLine();
          if("pea soup".equals(item)) {
               seenPeas = true;
          } else if("pancakes".equals(item)) {
               seenPancakes = true;
          }
     }

     if(seenPeas & seenPancakes) {
          System.out.println(name);
          break;
     }
}
System.out.println("Anywhere is fine I guess");
```

`break` will still run the rest of the code **after** the loop!

Non-Buffered IO & Large Inputs/Outputs

- `Scanner`
  - Easy to use, but is quite slow (due to use of regexes)
- `System.out.print*`
  - Will immediately give the value to the OS to display (i.e. unbuffered)
  - May use up a lot of time if called repeatedly

Buffered IO

- Faster but more complicated IO methods exist
- Some take-home assignment requires buffered/“fast” IO
  - Using `Scanner/System.out` will result in exceeding time limit
  - Rough rule of thumb:
    - If you are reading/writing in 10^5-10^6 words/characters/things
    - Then you probably want fast IO

BufferedReader API

- Provides a more efficient way for reading input (input **buffering**)
  - Non-buffered:
    - Every time you read some short word
    - Request from OS, for one short chunk each time
  - Buffered:
    - Request from OS, one large chunk/**buffer** at once
    - Slice it up word by word when needed

BufferedReader API

- Found in `java.io` package, need to use following line to import

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
```

- Declare a new `BufferedReader` object in main method

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

Wraps `System.in`, to provide buffering functionality.

- Read in input using methods found in `BufferedReader`

BufferedReader API

Method Signature	Description	Runtime
`String readLine()`	Reads a line of text. (Reads until it reaches the end of line.)	O(N)

Other methods exist but may not be as useful.

PrintWriter/BufferedWriter API

- Provides a faster way for writing output
- Found in `java.io` package, need to use following line to import

```
import java.io.PrintWriter;
import java.io.BufferedWriter;
import java.io.OutputStreamWriter;
```

- Declare a new `PrintWriter` object in main method

```
   PrintWriter pw = new PrintWriter(new BufferedWriter(
       new OutputStreamWriter(System.out) ));
```

Wraps `System.out`, to provide buffering and printing functionalities.

PrintWriter API

Method Signature	Description	Runtime
`void print(String s)`	Prints a string.	O(N)
`void println(String s)`	Prints a string and then terminates the line (with '\n').	O(N)
`void printf(String s)`	(Emulates the function in C programming language.)	O(N)
`void flush()`	Flush the stream. (Prints the current content of the writer to the screen)	O(1)
`void close()`	Closes the stream and releases any system resources associated with it. (Calls `flush()`, then closes the writer. The writer cannot be used again.)	O(1)

Slides covering API will cover the most frequently used (but not all) methods of a class.

PrintWriter/BufferedWriter API

- Used the same way as System.out
  - Delays printing until a `flush()` or `close()` method is called
  - Avoid repeated switching between printing and computation
  - Saves time
- **Always call `flush()` or `close()` on the PrintWriter before exiting your program**
  - If not, some output may not be printed

Safe Flushing/Closing

```
/* Unsafe (may lose last part of output) */

PrintWriter out = new PrintWriter(
    new BufferedWriter(
        new OutputStreamWriter(System.out)
    )));

// Program code...
out.close();
```

```
/* Try-with-resources */

try(PrintWriter out = new PrintWriter(
    new BufferedWriter(
        new OutputStreamWriter(System.out)
    ))) {
    // Program code...
}
```

```
/* Try-finally */

PrintWriter out = null;
try {
    out = new PrintWriter(new BufferedWriter(
        new OutputStreamWriter(System.out)
    ));

    // Program code...
} finally {
    if(out != null) {
        out.close();
    }
}
```

Kattio

- Kattis provides its own version of a buffered IO
    - Pre-packages all the stuff in the previous slides
- For input, it provides its own methods (next slide)
- For output, it uses the same methods as PrintWriter (previous slide)
    - Remember to flush/close at end!
- Available at https://github.com/kattis/kattio
    - If used, submit **both** your source code and Kattio source code.

Kattio.java API

Method Signature	Description	Runtime
`int getInt()`	Reads the next token in the input as an integer	O(N)
`long getLong()`	Reads the next token in the input as a long	O(N)
`double getDouble()`	Reads the next token in the input as a double	O(N)
`String getWord()`	Reads the next token in the input as a string	O(N)

Output methods are inherited from PrintWriter.

Safe Flushing/Closing (Kattio)

```java
/* Unsafe (may lose last part of output) */

Kattio io = new Kattio(System.in);

// Program code…

out.close();
```

```java
/* Try-with-resources */

try(Kattio io = new Kattio(System.in)) {
    // Program code...
}
```

```java
/* Try-finally */

Kattio io = null;
try {
    io = new Kattio(System.in)

    // Program code...
} finally {
    if(io != null) {
        io.close();
    }
}
```

Building a String

- Avoid repeated modifications (append/substrings) to Strings
  - e.g. Building a string in a loop
  - Java Strings are immutable
  - Every “modifying” operation has to allocate a new copy
    O(new_string_length) in both time/space
- If done in loop, can degrade to O(n^2)!

```
String s = "";
for(int i=0; i<n; i++) {
  s = s + "hello ";
}

T(n) = 6 + 12 + 18 + … 6n
     = 6(n^2+n)/2 ∈ O(n^2)
```

StringBuilder

- Solution (for repeated appends): Use StringBuilder
  - StringBuilders can be modified, without needing to reallocate
  - Only need to freeze at end (when converting toString())

```
String s = "";
for(int i=0; i<n; i++) {
  s = s + "hello ";
}     O(new_length) = O(i)


T(n) = 6 + 12 + 18 + … 6n
     = 6(n^2+n)/2 ∈ O(n^2)
```

```
StringBuilder s = new StringBuilder();
for(int i=0; i<n; i++) {
  s = s.append("hello ");
}     O(hello_length) = O(1)
String s = sb.toString();

T(n) = 6 + 6 + 6 + … + 6
     = 6n ∈ O(n)
```

StringBuilder

- Represents a mutable (modifiable) buffer of characters
  - i.e. a mutable `String`.
  - Cheap to append at rear
    - O(added_string_length) for `StringBuilder.append`

      O(new_string_length) for `String.concat` or `+`
  - Also cheap to delete near rear

Extra Note: Some of you may have seen `StringBuffer`.

It is a supposedly more “threadsafe” version of `StringBuilder`, the only difference being all methods are `synchronized`.

However, that isn’t actually useful for sequences of append operations, and so `StringBuffer` is effectively deprecated.

StringBuilder

Method Signature	Description	Runtime
`char charAt(int i)`	Returns the character at index i (0-based)	O(1)
`int length()`	Returns length of current string	O(1)
`StringBuilder append (String s)`	Adds `s` to the back of the stored string. Returns `this` for method chaining. (This method has various overloads for `int, char, Object`, etc.)	O(|s|) (amortized)
`String substring (int start, int end)`	Creates a new **immutable** string, with the current contents, in the range [start,end), in 0-based indexing. Similar to `String.substring(int, int)`.	O(end-start)
`String toString()`	Creates a new **immutable** string with the current contents. (Overrides `Object.toString()`.)	O(N)

Other possibly useful methods: `delete(int, int), deleteCharAt(int), reverse()`

StringBuilder

- Suppose we have an array of Strings
  - We want to add a line number to each of them
  - Then join them into a single String

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

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

Note the use of **method chaining**:
`sb.append(...)` returns `sb` itself, so you can call more append methods.

“Just Print It Out”

- Solution (if directly printing output):
    - Immediately print out items, instead of building an output String
    - `System.out/BufferedWriter` will handle all of it for you
- Not always applicable.
    - Only if no need to manipulate the string further

Take Home 1A - Train Passengers

- Given the train capacity and list of stations which it visits
- Determine if there are any inconsistencies:
  a. Train has more passengers than its capacity will allow
  b. Train falls below 0 passengers
  c. Train is not full, but passengers decide to wait at the station
  d. Train is not empty at the last station
  e. Passengers got on or waited at the last station

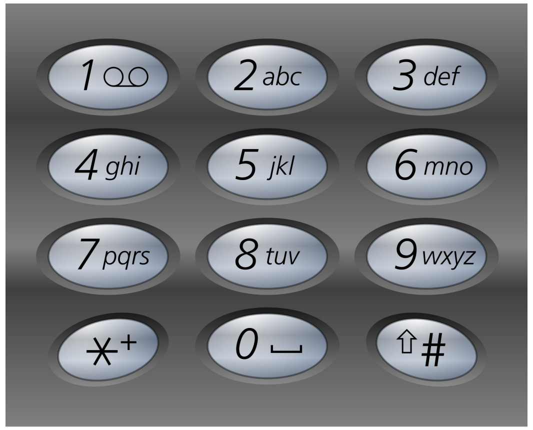
Take Home 1B - Best Relay Team

- Given a list of runners and their times as the first/subsequent runner
  - Find team arrangement resulting in shortest total time
- Trying all possible $O(N^4)$ teams will take too long
  - Back of envelope: $500^4 = 6.25 \times 10^{10} >> 10^8$ ops limit
- Find all choices amongst a smaller subset of runners?
  - Hint: Which runners should we **even bother considering**?
    - For each type of slot

Take Home Lab FAQ

- You will not get bonus marks for optimizing your code
- Future take home labs may contain bonus questions for you to try
- You will not be penalized for code styling

One-day 1 - T9 Spelling



- Simulate typing on old phone keypads
- Each letter has different keypress sequence
    - a -> 2
    - b -> 22
    - s -> 7777
- Given a string of letters, how to type it?
    - “as” -> “27777”
    - “aa” -> “2␣2”
        - Need pause, otherwise becomes “b”

One-day 1 - T9 Spelling

- Given some letter,
    - How to determine corresponding keypress sequence?
- Given a string of letters,
    - When should I insert a pause?
    - How to efficiently combine into final answer?

One-day 1 - T9 Spelling

- Every ASCII character is represented as an integer from 0 to 127
  - Known as its ASCII value
- Possible to simulate a dictionary (for Python / Javascript users)
  - Create an array of length 128
  - Use the character as the index

```
int[] arr = new int[128];
String input = "cd";
arr['c'] = 123;
char letter = input.charAt(0); // letter = 'c'
System.out.println(arr[letter]); // Prints 123
```

One-day 1 - T9 Spelling

- Provided sample input for this question covers most of the special cases
- What about possible cases that are not covered?
    - Must be legal input, given input format and constraints

One-day 1 - T9 Spelling

- 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

- Include your **name** and **student number** in comments at the top of your code.
- You are allowed (and encouraged) to discuss algorithms
  - List down all your collaborators in your source code
- **You are NOT allowed to:**
  - **Copy another person’s code**
  - **Look at another person’s code**
  - **Use another person’s code as a base for your own code**
- Plagiarism checks will be in place

Extras for Take Home Labs

Sorting in Java

- `Arrays.sort(arr)` will sort an array (e.g. `int[], String[]`)
  - For primitive types (`int[]`, `float[]`, etc)
    - Dual-pivot Quicksort (by Vladimir Yaroslavskiy, Jon Bentley, and Joshua Bloch)
      - If ints/floats/etc compare equal, they have the **exact** same value
      - No need to worry about stability
  - For reference types (`Integer[]`, `String[]`, etc.)
    - Adaptive Mergesort (based off Timsort)

Sorting in Java

- `Collections.sort(myList)`
  - Sorts a `List` (from Java API) of objects
  - Implementation: Adaptive Mergesort/Timsort
  - Also available as `myList.sort()`

Sorting in Java

- Before using the sorting API, **at least one of the following** must be done:
    - Objects being stored in the list implements the `Comparable` interface (referred to as the natural order)
    - `Comparator` is provided for the sorting method
- Sorting in Java API will handle swapping of elements automatically
    - Just need to provide a method for ordering the elements

Arrays API

All time complexities here assume the Comparator/Comparable methods are O(1).

Method Signature	Description	Runtime
`void sort(Object[] a)`	Sorts the specified array of objects into ascending order, according to the natural ordering of its elements. All objects must be the same type and implement Comparable.	O(N lg N)
`void sort(Object[] a, int fromIndex, int toIndex)`	Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements. All objects must be the same type and implement Comparable.	O(N lg N), where N = range
`void sort(T[] a, Comparator<? super T> c)`	Sorts the specified array of objects according to the order induced by the specified comparator. (Sorts the array using the provided `Comparator`)	O(N lg N)
`void sort(T[] a, Comparator<? super T> c, int fromIndex, int toIndex)`	Sorts the specified range of the specified array of objects according to the order induced by the specified comparator.	O(N lg N), where N = range

Collections API

All time complexities here assume the Comparator/Comparable methods are O(1).

Method Signature	Description	Runtime
`void sort(List<T> list)`	Sorts the specified list into ascending order, according to the natural ordering of its elements.	O(N lg N)
`void sort(List<T> list, Comparator<? super T> c)`	Sorts the specified list according to the order induced by the specified comparator.	O(N lg N)

You can use the subList() method of a list, to create a subrange to be sorted.

```
// Sorts the range [5, 10) in myList
Collections.sort(myList.subList(5, 10));
```

Comparable Interface

- A class with Comparable interface does not require comparator to be used when sorting

```
class className implements Comparable<classToCompareWith> {
...
}

class Runner implements Comparable<Runner> {
...
}
```

Comparable Interface

- A class with Comparable interface MUST implement compareTo method
- Compares the current object with the other object given
    - Returns negative value if current object is smaller than the other object
    - Returns 0 if they are the same
    - Returns positive value if current object is larger

Comparable Interface

- A class with Comparable interface MUST implement compareTo method

```
class Runner implements Comparable<Runner> {
    double time;
    @Override
    public int compareTo(Runner other) {
        return Double.compare(this.time, other.time);
    }
}
```

Note that most standard classes have its own .compare/compareTo method!

Comparator<T> API

Method Signature	Description	Runtime
`int compare(T left, T right)`	Compares the two given objects. ● `left < right` : Return negative ● `left == right` : Return 0 ● `left > right` : Return positive	User-imple mented

Tedious to implement compare() by hand, especially when handling tiebreaks!

Comparator<T> Helper API: Extract/Tiebreak

Method Signature	Description	Runtime

`static <T, U extends Comparable<U>>`
`Comparator<T> comparing(`
`  Function<T, U> extractorFn)`
	Given a function to **extract** a "key" (of type U) out of a value of type T, create a Comparator<T> that compares on the corresponding **extracted** U "keys".	O(1)

`default Comparator<T> thenComparing(`
`  Comparator<T> other)`
	Returns a **new** comparator, that tries `this` comparator first, then tiebreaks with the `other` comparator.	O(1)

`default <U extends Comparable<U>>`
`Comparator<T> thenComparing(`
`  Function<T, U> extractorFn)`
	Equivalent to:
	`this.thenComparing(Comparator.comparing(extractorFunction))`
		O(1)

Generic signatures are simplified slightly. Full signatures are available in Javadocs.

Primitive variants of comparing/thenComparing are also available but not listed.

Comparator<T> Helper API: Extract/Tiebreak

```
List<Integer> list = new ArrayList<>(List.of(12, 31, 40,
79, 82));

Comparator<Integer> comp1 = Comparator.comparingInt(
    value -> value % 10); // Extract last digit
Collections.sort(list, comp1);
// list is now [40, 31, 12, 82, 79]

Comparator<Integer> comp2 = comp1.thenComparingInt(
   value -> Math.abs(value-100));// Extract distance to 100
Collections.sort(list, comp2);
// list is now [40, 31, 82, 12, 79]
```

Comparator<T> Helper API: Reversing

Method Signature	Description	Runtime
`static <T extends Comparable<T>> Comparator<T> reverseOrder()`	Returns a Comparator that uses the reverse of T's natural order.	
`default Comparator<T> reversed()`	Returns a Comparator with reverse order to `this` Comparator.	User-implemented

Generic signatures are simplified slightly. Full signatures are available in Javadocs.

Comparator<T> Helper API: Reversing

```
List<Integer> list = List.of(1,6,7,2,9,4,3,8,5);

Collections.sort(list);
// list is now [1,2,3,4,5,6,7,8,9]

Collections.sort(list, Comparator.reverseOrder())
// list is now [9,8,7,6,5,4,3,2,1]
```