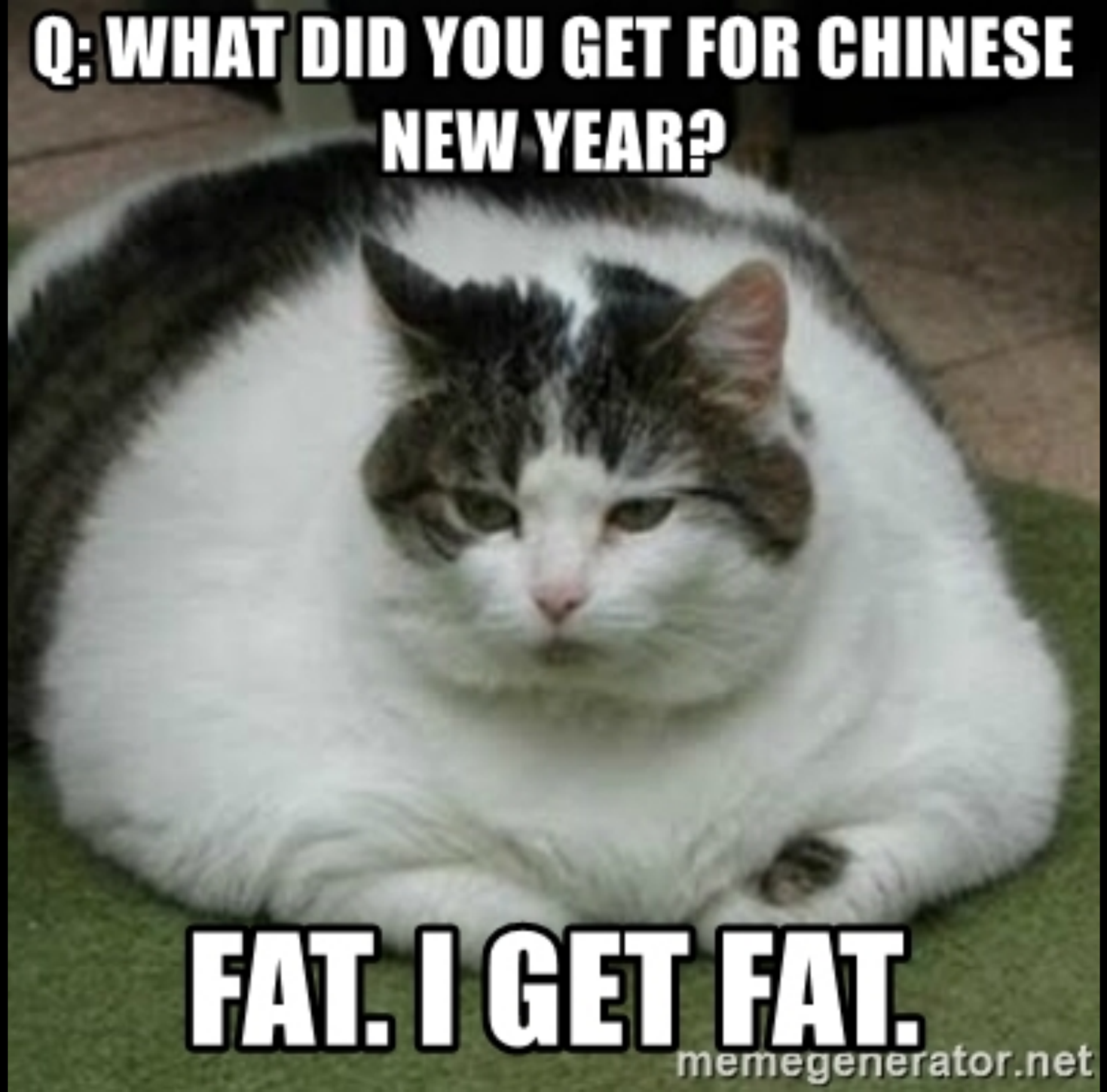
Week 4 Lab Session

CS2030S AY21/22 Semester 2

Lab 14B

3 Feb 2022

Yan Xiaozhi (David)
@david_eom
yan_xiaozhi@u.nus.edu


Q: WHAT DID YOU GET FOR CHINESE NEW YEAR?
FAT. I GET FAT.
memegenerator.net

Admin

- Contact tracing & QR code
- Due to CNY holidays:
  - Lab 1 ungraded
  - Lab 1 & Lab 2 will be graded together
  - Comments from me are just feedbacks
- Lab 2 deadline: 8 Feb (Tue) (11 Feb?)

`vimrc` Setup

- Configuration file for your vim
- Located in home directory i.e. `~/.vimrc`
- Please experiment with it
  - Comes in really handy for PEs
  - Help you achieve good coding style
- `gg=G` to automatically indent all the code in the current file (according to your config)

```
 1 set number
 2 set tabstop=4
 3 set shiftwidth=4
 4 set autoindent
 5 set smartindent
 6 set expandtab
 7 set linebreak
 8 set wrap
 9 set showcmd
10 set backspace=indent,eol,start
11 syntax enable
12 inoremap ( ()<Left>
13 inoremap [ []<Left>
14 inoremap { {}<Left>
15 inoremap ' ''<Left>
16 inoremap " ""<Left>
17 inoremap {<CR> {<CR>}<Esc>O
```

Week 3 Content Recap

- Abstract class
- Interface
- Wrapper class

Lab 1 Feedback

Single Responsibility Principle

- Part of the SOLID Principles (an acronym made up of the first letter of each principle)
  - Single responsibility, Open-closed, Liskov substitution (already learnt), Interface segregation, Dependency inversion
- “A class should have one, and only one, reason to change.”
- A class should only have one job
- Major violation in the original skeleton code
- Polymorphism, when applied correctly, should not have a class take on multiple responsibilities

Polymorphism

- Most students got the idea of polymorphism right
- When applied correctly, complex `if/else` statements are avoided
- Split ShopEvent into 4 sub classes (kind of a no brainer)
  - `ArrivalEvent`
  - `ServiceBeginEvent`
  - `ServiceEndEvent`
  - `DepartureEvent`
- Naming?

Information Hiding

- Obeyed by most students, careless slip for some of the class fields
- All object variables should be set to `private` unless absolutely necessary
- `protected` is NOT sufficient!!

Encapsulation

- Significant amount of violations
- Encapsulate information into its own class!!
- Logic related to a specific class should exist in that class only

Styling

- Will be penalised in subsequent labs!
- Follow the style guide in textbook
- Run checkstyle before submission
  - Put command in bash script!
  - `~/.bash_profile`
  - `alias checkstyle='<actual command>'`
  - `source .bash_profile`

```java
public ServiceEnd(int eventType, double time, int customerId,
                  double serviceTime,  int counterId, boolean[] available){
        super(time);
        this.eventType= eventType;
        this.customerId = customerId;
        this.serviceTime= serviceTime;
        this.available = available;
        this.counterId = counterId;

}
```

CS2030/S Java Style Guide

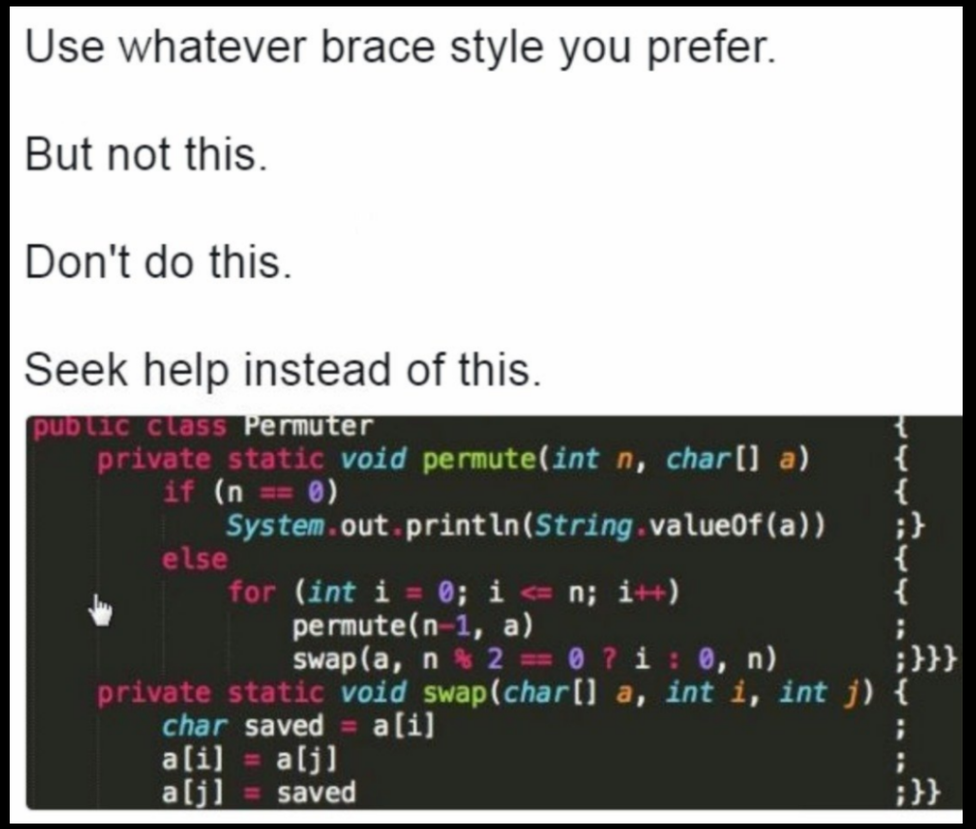
Why Coding Style is Important

One of the goals of CS2030/S is to move you away from the mindset that you are writing code that you will discard after it is done (e.g., in CS1101S missions) and you are writing code that noone else will read except you and your tutor. CS2030/S prepares you to work in a software engineering teams in many ways, and one of the ways is to enforce a consistent coding style.

If everyone on the team follows the same style, the intend of the programmer can become clear (e.g., is this a class or a field?), the code is more readable and less bug prone (e.g., the Apple `goto fail` bug). Empirical studies support this:

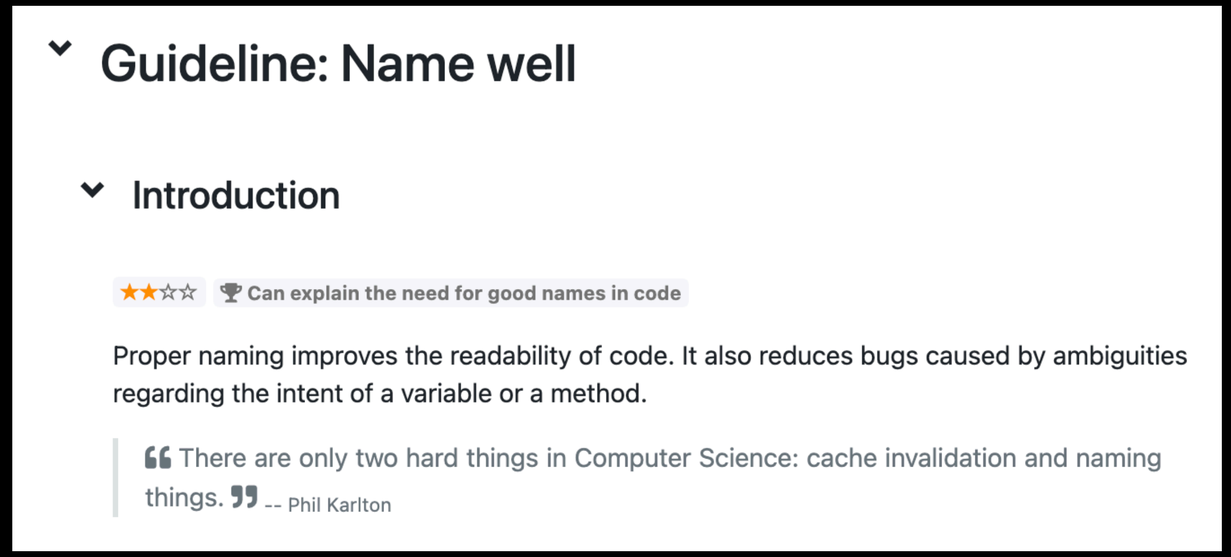
Styling

- Why is styling important?
  - Bopes, this is a requirement by teaching team LOL
  - Emphasised in future modules as well e.g. CS2103T, CS3203, etc.
  - Communication, your code will be read by other engineers
  - Companies also have standards: https://google.github.io/styleguide/javaguide.html
  - Makes it easier for me to grade heh :)



Others

- Use better variable and class names in general
- https://nus-cs2103-ay2122s1.github.io/website/se-book-adapted/chapters/codeQuality.html#guideline-name-well
- Unused variables
- Multiple copies of the same object being instantiated



```
1  class ServiceEnd extends Event {
2
3    private int customerId;
4    private int counterId;
5    private double serviceTime;
6    private Counter[] counters;
7
8    public ServiceEnd(double time, int customerId, double serviceTime,
9        int counterId, Counter[] counters) {
10     super(time);
11     this.customerId = customerId;
12     this.serviceTime = serviceTime;
13     this.counterId = counterId;
14     this.counters = counters;
15   }
```

OOP Modelling Exercise

- The COVID-19 Task Force is developing a system to keep track of confirmed COVID-19 cases in Singapore. Each confirmed case has an integer case id. There are two types of confirmed cases: imported and local. For each imported case (and only imported case), the system keeps track of the country the case is imported from. For each confirmed case, the system keeps track of the contacts of the case. A contact is another confirmed case (can be either local or imported). A case can have zero or more contacts. Each contact is labelled with the nature of the contact, which can be either one of the three: casual contact, close contact, family member.
- The cases can be grouped into clusters. Each cluster has a name (a String). A cluster can contain one or more cases. But a case might not necessarily belong to a cluster. A case can belong to multiple clusters.
- Important operations are:
- Given a cluster, find all cases in the cluster.
- Given a cluster, find all important cases in the cluster.
- Given a case, find all close contact of the case.

Lab 2 Overview

Before you start…

- Refactor your lab 1 code first
- This will make your life much easier
- Do feel free to ask me if you do not understand my comment!
- Follow the instructions in `Questions.md` and copy your rectified code for lab 2

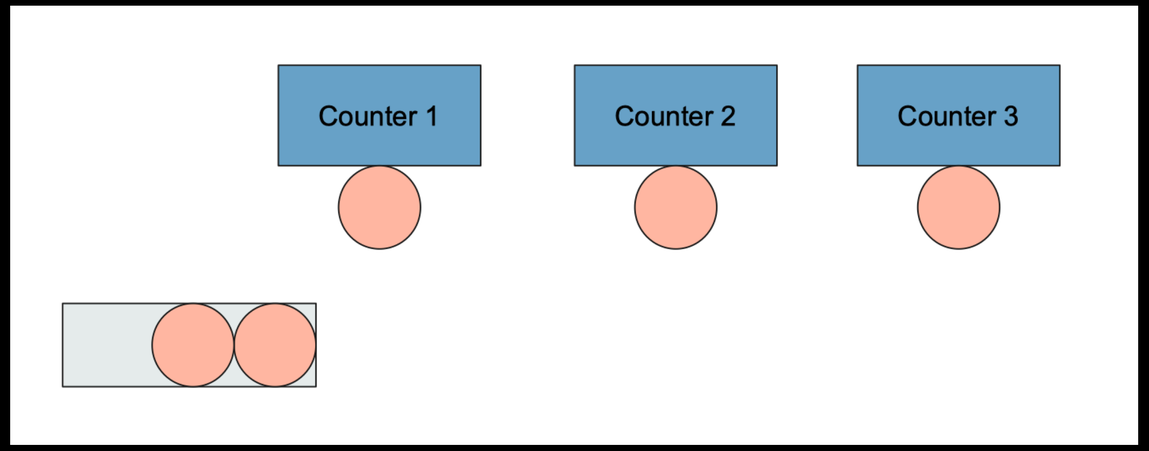
Lab 2: Discrete Event Simulator (Part 2)

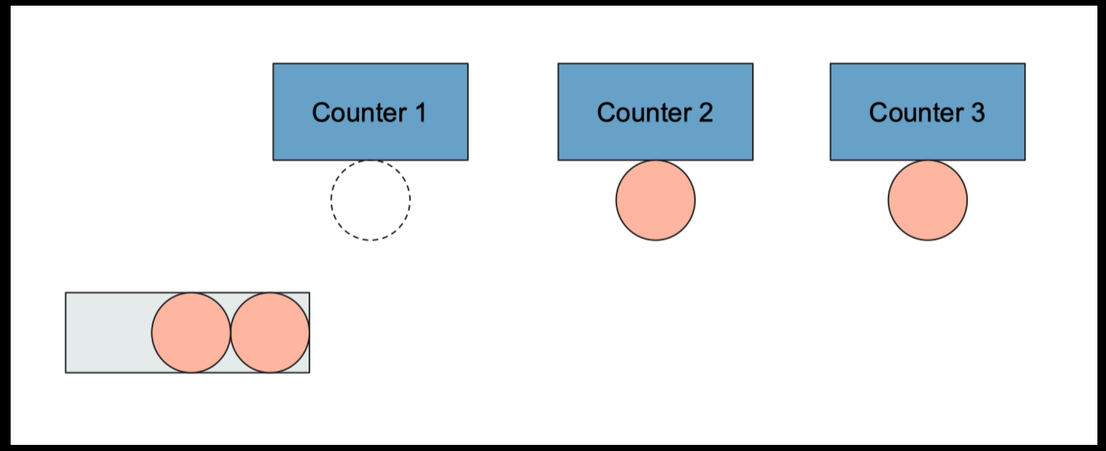
- Requirements
  - Add an entrance queue of max size `m`
  - Read input for max queue length
  - Various printing modifications
    - “C1” instead of “Customer 1”
    - “S1” instead of “Counter 1”
    - Printing of queue during arrival event and customer joining queue

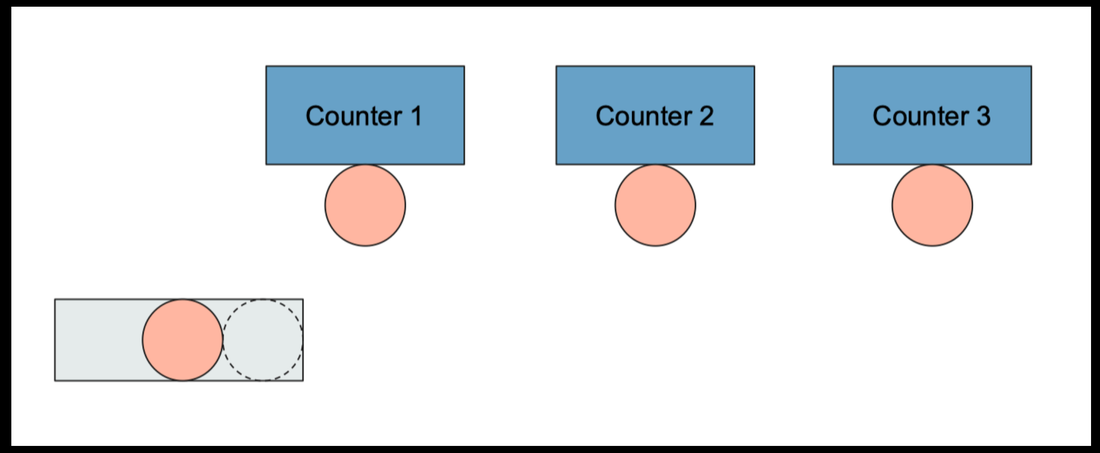
Lab 2: Discrete Event Simulator (Part 2)

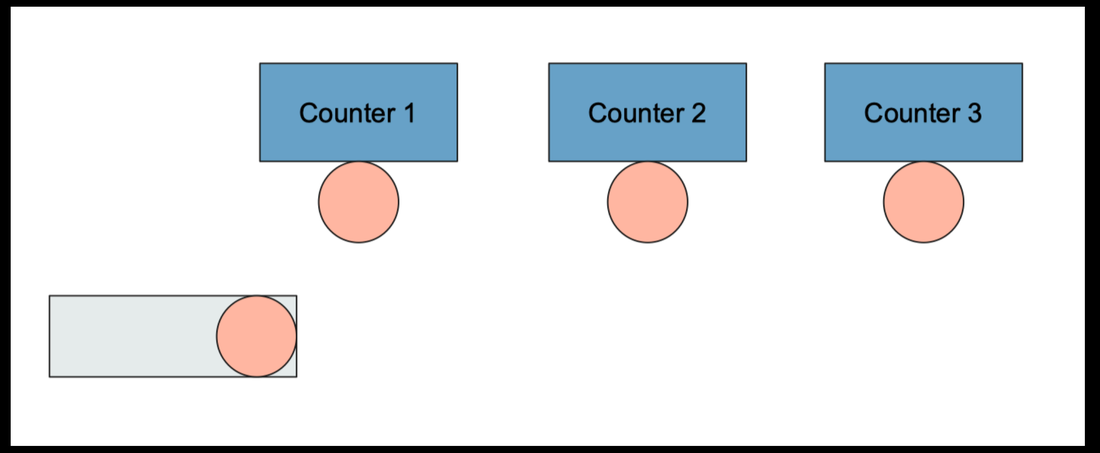
- Updated workflow
  - 1. Customer arrives
  - 2. If a counter is available, serve customer
  - 3. Else if entrance queue is not full, customer waits in queue
  - 4. Else, customer departs the shop
- When counter is done with serving a customer, the first earliest customer in the queue gets served

Lab 2: Discrete Event Simulator (Part 2)









Lab 2: Discrete Event Simulator (Part 2)

- What is `Queue` class and how is it implemented?
- Simply do not need to care!
- Just use it as described, similar to Java API

```
// Create a queue that holds up to 4 elements
Queue q = new Queue(4);

// Add a string into the queue.  returns true if successful;
// false otherwise.
boolean b = q.enq("a1");

// Remove a string from the queue.  `Queue::deq` returns an
// `Object`, so narrowing type conversion is needed.  Returns
// `null` is queue is empty.
String s = (String) q.deq();

// Returns the string representation of the queue (showing
// each element)
String s = q.toString();

// Returns true if the queue is full, false otherwise.
boolean b = q.isFull();

// Returns true if the queue is empty, false otherwise.
boolean b = q.isEmpty();

// Returns the number of objects in the queue
int l = q.length();
```

Lab 2: Discrete Event Simulator (Part 2)

- If you are changing too much code at too many places, it is probably an indication that you are not using OOP
- Styling issues & warnings WILL be penalised
- Do NOT use git directly to touch the repo, 50% penalty applicable

Happy coding!