Week 5 Lab Session

CS2030S AY21/22 Semester 2

Lab 14B

10 Feb 2022

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

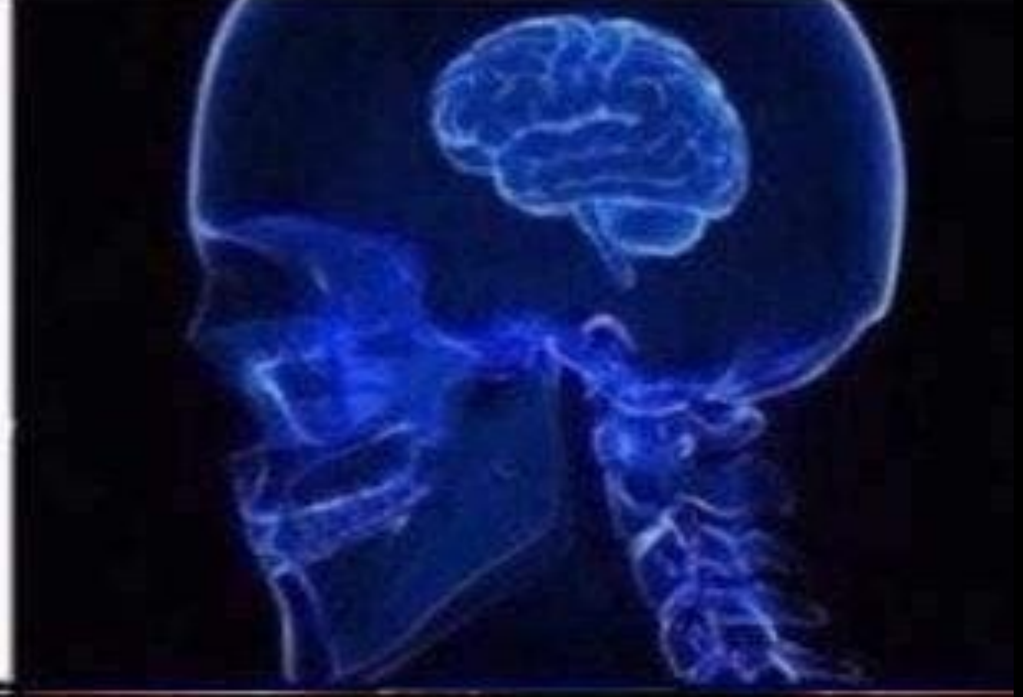
Admin

- Contact tracing & QR code
- Lab 2 grading still in the progress, will try to finish by end of week
- Telegram group chat! bit.ly/lab14bfeedback
- If you feel unwell:
  - Drop me a message if you would like to attend class online

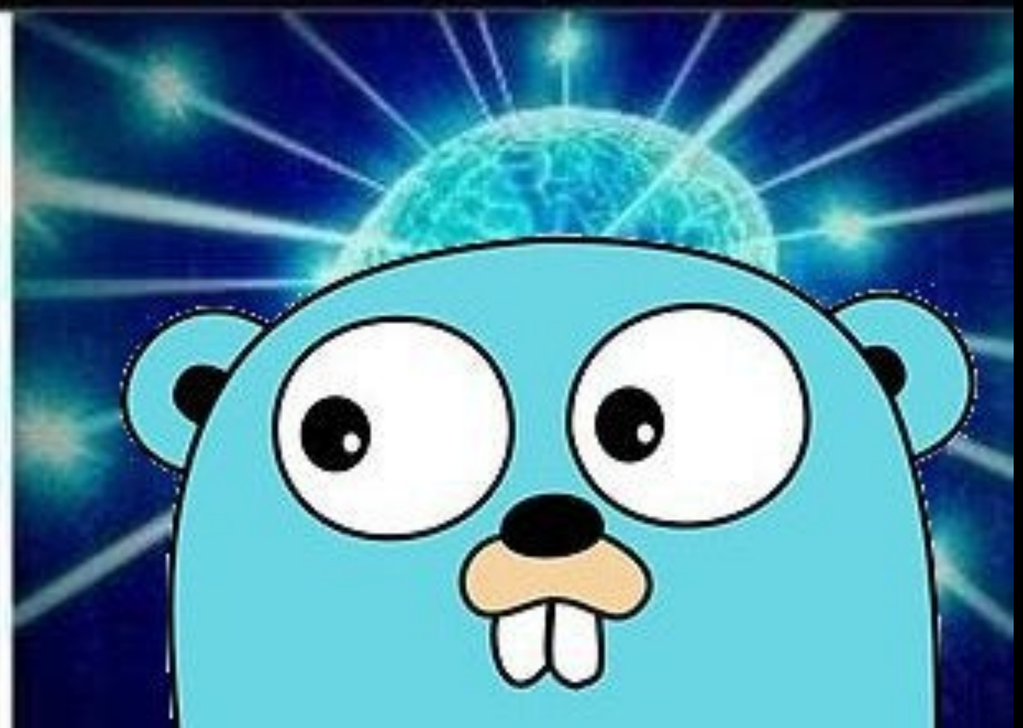
`vimrc` Setup

- `gg=G` does not work?
  - piazza.com/class/kwx6svvqrhw3qm?cid=227
  - `filetype plugin indent on`
  - Do not follow blindly
  - Experiment as much as possible on your own!

```
template<typename T>
T min(T a, T b)
{
    return a < b ? a : b;
}
```



manually rewriting the same function for every type there is



Week 4 Content Recap

Generics

- Parameters that takes in types
- Declare a generic class with type parameter `T` and a single `private` field `x` of type `T`
- Instantiate an instance of `A<T>` with type argument Integer
- Declare a generic class `B<T>` that extends from `A<T>`, and a generic class `C<T>` that contains a field of type `A<T>`
  - Are the occurrences of `T` refers to the same `T`?
- What's wrong with `class F extends A<T> { }`?
- What about `class F extends A<String> { }`?

Generics

- Exercise: write a generic method that copy from one array to another
- ```
  class A {
    public static ??? void copy(??? from, ??? to) {
      for (int i = 0; i < from.length; i++) {
        to[i] = from[i];
      }
    }
  }
  ```
- ```
  String s[] = new String[2]; String t[] = new String[2];
  Integer i[] = new Integer[2]; Integer j[] = new Integer[2];
  ```
- ```
  A.<String>copy(s, t); A.<Integer>copy(i, j); // ok
  A.<String>copy(i, j); A.<String>copy(s, j);  // error
  ```
  ```

Generics

- Do NOT use raw types e.g. `new A()`
  - Major sources of mark deductions in labs and PE!
- Type inference e.g. `new A<>()` is NOT encouraged at this juncture
- Always use `-Xlint:rawtypes` flag from now on
  - Alias?

Type Erasure

- Honestly, why?
  - Compatibility with the older version of Java
  - And many other reasons e.g. parametricity
- If type unbounded: Object
- If type bounded: bound
- If multiple bounds: first bound
  - `class MyList<T extends Serializable & Comparable>`
- If wildcard…?: later on in the module!

Unchecked Warnings

- Design a generic class `D<T>` that contains a field that is an array of type `T` with 10 elements, instantiate that array in the constructor
- Write a generic class `E<T extends Comparable<T>>` that contains a field that is an array of type T with 10 elements. Instantiate that array in the constructor
- ```
  class E<T extends Comparable<T>> {
    T[] a;
    E() {
      @SuppressWarnings({"unchecked", "rawtypes"})
      T[] tmp = (T[]) new Comparable[10];
      this.a = tmp;
    }
  }
  ```
  ```

Exceptions

- Will this compile?
- ```
  import java.io.File;
  import java.util.Scanner;

  class ExceptionDemo {
    public static void main(String[] args) {
      File f = new File("hello.txt");
      Scanner s = new Scanner(f);
    }
  }
  ```
  ```

Exceptions

```
import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;

class ExceptionDemo {
  public static void main(String[] args) {
    File f = new File("hello.txt");
    try {
      Scanner s = new Scanner(f);
    } catch (FileNotFoundException e) {
      // do something
    }
  }
}
```

Exceptions

- What about this?
- ```
  import java.io.File;
  import java.util.Scanner;

  class ExceptionDemo {
    public static Scanner openFile(String filename) {
      File f = new File(filename);
      return new Scanner(f);
    }

    public static void main(String[] args) {
      Scanner sc = openFile("hello.txt");
    }
  }
  ```
  ```

Solution 1: catch in `openFile`

```
import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;

class ExceptionDemo {
  public static Scanner openFile(String filename) {
    File f = new File(filename);
    try {
      return new Scanner(f);
    } catch (FileNotFoundException e) {
      return null;
    }
  }

  public static void main(String[] args) {
    Scanner sc = openFile(“hello.txt"); // need to handle null afterwards
  }
}
```

Solution 2: pass on to `main`

```
import java.io.File;
import java.util.Scanner;
import java.io.FileNotFoundException;

class ExceptionDemo {
  public static Scanner openFile(String filename) throws
FileNotFoundException{
    File f = new File(filename);
    return new Scanner(f);
  }

  public static void main(String[] args) {
    try {
      Scanner sc = openFile(“hello.txt");
    } catch (FileNotFoundException e) {
      // do something
    }
  }
}
```

Lab 1 & 2 Feedback

Grading Scheme

- Correctness - 2 marks
- OO Design - 20 marks
  - Information Hiding - 4 marks
  - Encapsulation - 4 marks
  - Tell, Don't Ask - 4 marks
  - Inheritance/Polymorphism - 8 marks
- Compilation error

Information Hiding

- All object variables set to `private`
- -1 for each violation

Encapsulation

- Minimally, the following classes are to be created (naming can be different)
  - `Shop`
    Encapsulates the counters and the entrance queue
    Look for available counters
  - `Counter`
    Encapsulates the availability and the counter ID
  - `Customer`
    Encapsulates the customer ID, service time, arrival time
    Look for counters to join
- -1 for each missing encapsulation

Tell, Don’t Ask

- Finding available counters done by `Shop`
- `Customer`, `Counter` to be responsible for its own string representation
- Getters will be scrutinised
- Checking if the queue is full:
  - `Shop::isQueueFull()` is okay, albeit boilerplate code
  - Still better than passing the whole queue to let others find out themselves
- -1 for each violation

Inheritance/Polymorphism

- `ShopEvent` broken into at least five subclasses
- `toString` and `simulate` should override Event's methods
- No `if-else` conditional check or `instanceof` check for `eventType`
- -1 for each missing new class
- -1 for not overriding
- -1 for `if-else` block

Others

- Use of `this`
- Overriding methods `@Override`
- Creation of `ServiceBeginEvent` for next customer
  - `ServiceEndEvent` vs `DepartureEvent`
- Unused variables/parameters!

Lab 3 Overview

Lab 3: Discrete Event Simulator (Part 3)

- Requirements
  - Each counter has a queue + entrance queue
    - Additional input parameter, think about encapsulation!
  - Make `Queue` generic
  - `Counter` should be comparable with itself
    - `counters.min()` should return the next counter to join
  - Create generic `Array` class to hold `Counter`
  - Print statement adjustments

Some Reminders

- Don’t be last minute! (trust me it ain’t fun…)
- Run checkstyle before submission
- 4% lab

Happy coding!