

# Week 6 Lab Session

CS2030S AY21/22 Semester 2

Lab 14B

17 Feb 2022

Yan Xiaozhi (David)  
@david\_eom  
[yan\\_xiaozhi@u.nus.edu](mailto:yan_xiaozhi@u.nus.edu)

# Admin

- Contact tracing & QR code
- Lab 3 grading in the process
  - I know prof said all feedbacks will be given by Thursday lab...
  - Not related to Lab 4, chill!
- Late submission policy (I can't control!)

# Lab 1 - 3 Review

# So... WTH happened?

- Discrete event simulator
- Originally a semester-long project in CS2030
- Repurposed and shrank down into a 3-week project in CS2030S
- Removed a lot of requirements / features in the current version
  - Think about how you can implement them!

# What Have You Missed Out?

- Different types of **Customer**
  - Normal customer: joins the first counter queue sorted by ID
  - Greedy customer: joins the counter with shortest queue
- Different types of **Counter**
  - Human-served counters: takes a break after serving every customer
  - Self-service counters: serves the next customer straightaway

# What Have You Missed Out?

- Random seed
  - Pseudo-RNG class to simulate probability
  - Customers and their arrival time determined by random seed
  - Customers have a probability to be greedy or normal
  - Human-served counters have a probability to rest
- FuNcTiOnAl PrOgRaMmInG
  - No loops/recursions, all logic using lambda expressions
  - Prepare for this after recess week :)

# Looking Back...

- Lab 1 is probably one of the hardest
  - Your first exposure to OOP
  - Using vim
  - Modifying existing code (brownfield project) instead of starting afresh
  - Chugging a few hundred lines of code at one shot
  - ... but this is the time when you truly learn something!





# About Designing Code

- Strict requirements
- Well-designed code != less code
- Boilerplate may be needed to achieve it
- There is no “absolute” answer in code design
  - SOLID
  - Other design principles

In this module,

We still want a good product.  
But we take a longer path get there.

○ product → tools/techniques/process → you 

○ you → tools/techniques/processes → product 

In a real project, **don't prioritize tools/techniques/processes** over the product!



# Week 6 Content Recap

- Wildcards
  - Upper-bounded / lower-bounded / unbounded
  - Remember PECS!!!
- Type Inference
  - Diamond operator
  - Normal inference
  - Target inference

# PECS

- `public Number sum(Collections<? extends Number> arr) {  
 // sums up all the values in the collection  
}`
- `Integer? Long? Double? Float?`
- `ArrayList<Number> arr = new ArrayList<Number>();  
public void copyTo(Collections<? super Number> arr) {  
 // Copies current array to the Collection  
}`
- `Collections<Number>? Collections<Object>?`

**Before We Dive Into Lab 4...**

**I tried to solve a problem with  
Java**

**I now have a ProblemFactory**

# Factory Method

- Create objects without exposing the creation logic to the client
- Client uses the same common interface to create a new type of object
- Write a class `A` such that it behaves as follows:
- ```
jshell> new A()  
|   Error:  
|   A() has private access in A
```
- ```
jshell> A.construct()  
$ ==> A@26be92ad
```
- ```
jshell> A.construct()  
$ ==> A@224edc67
```



# Factory Method

- Change the construct method of `A` so that it always return the same instance
- `jshell> A.construct()`  
`$ ==> A@26be92ad`
- `jshell> A.construct()`  
`$ ==> A@26be92ad`



# Factory Method

- Change A so that the factory method takes in an int as parameter
  - When the argument is 0, the same instance is always returned
  - When the argument is non-zero, a new instance is always created
- ```
jshell> A.construct(0)
$ ==> A@26be92ad
jshell> A.construct(0)
$ ==> A@26be92ad
```
- ```
jshell> A.construct(1)
$ ==> A@224edc67
```
- ```
jshell> A.construct(1)
$ ==> A@4b9e13df
```



# Comparing Two Generics

- Write a generic class `B` with type parameter `T` and a single private field `x` of type `T`
- Override the `equals` method of `Object` to compare if two `Bs` are equals.

- ```
jshell> B<Integer> b = new B<>(4);  
b ==> B@26be92ad
```
- ```
jshell> b.equals(b)  
$ ==> true  
jshell> b.equals(new B<>(4))  
$ ==> true
```
- ```
jshell> b.equals(new B<String>("hello"))  
$ ==> false  
jshell> b.equals(new B<>(null))  
$ ==> false  
jshell> b.equals(null)  
$ ==> false
```

# Lab 4 Overview

**Module** [java.base](#)**Package** [java.util](#)

## Class **Optional**<T>

[java.lang.Object](#)[java.util.Optional](#)<T>

### Type Parameters:

T - the type of value

```
public final class Optional<T>
    extends Object
```

A container object which may or may not contain a non-null value. If a value is present, `isPresent()` returns true. If no value is present, the object is considered *empty* and `isPresent()` returns false.

Additional methods that depend on the presence or absence of a contained value are provided, such as `orElse()` (returns a default value if no value is present) and `ifPresent()` (performs an action if a value is present).

This is a [value-based](#) class; use of identity-sensitive operations (including reference equality (`==`), identity hash code, or synchronization) on instances of `Optional` may have unpredictable results and should be avoided.

# Lab 4: Box

- Build a generic wrapper class `Box<T>` to store an item of any type `T`
  - Contain a `private final` field of type `T` to store the item
  - Override `equals` method
  - Override `toString` method
  - Static method of that returns a box with the given method (factory method)
    - Keep constructor private!

# Lab 4: Box

- `EMPTY_BOX`
  - Cached and can be reused
  - `private final`
  - What type should `EMPTY_BOX` be?
- `ofNullable`
  - Behaves just like `of`, but returns an empty box when input is `null`

# Lab 4: Box

- Create interface `BooleanCondition<T>` with a single `abstract boolean` method `test`
- Create method `filter` in `Box` that takes in a `BooleanCondition`
  - Returns `EMPTY_BOX` if condition fails, or the original box if condition passes
- Create classes that extend from `BooleanCondition`:
  - `DivisibleBy`
  - `LongerThan`

# Lab 4: Box

- Create interface `Transformer<T, U>` with `abstract` method `transform` that takes in type `T` and returns type `U`
- Create method `map` in `box` that takes in a `Transformer`, uses `transform` to transform the box into another box of type `Box<U>`
- Create classes that extend from `Transformer`:
  - `LastDigitsOfHashCode`
  - `BoxIt<T>`



# Reminders

- NO RAW TYPES! -1 for each violation
- Use `@SuppressWarnings` responsibly, -1 for each
- Prepare for PE:
  - Read `Question.md` with vim
  - `:vsp` to split screen in vim
  - Only one maximised terminal window allowed during PE
- Panopto recording
  - [nus-cs2030s.github.io/2122-s2/panopto/panopto.html](https://nus-cs2030s.github.io/2122-s2/panopto/panopto.html)

Happy coding! 