CS2030S Recitation Problem Set 6

Brian

Recap

Maybe

- Just think of it as a box containing some value
- Ok but why do we need this box?
  - We want to abstract out `null` checks (absence of a value)
  - This absence would be captured by `None`
  - Use some APIs to work on the value
  - APIs would interally handle the `Some` / `None` cases
  - Chain these API calls to have look elegant

Maybe APIs

- `of`: Creates a `Maybe` containing our value (or `None` if given a `null`) You can think of this as "lifting" into the `Maybe` type.
- `map`: Takes a function and applies it on the value if `Some`, propogates if `None`
- `filter`: Similar to filter in CS1101S (if fail become `None` else remain the same)

More Maybe APIs

- `flatMap`: Takes in $f : X \to \text{Maybe<Y>}$ If `None` remains `None`, applies on x to produce $f(x)$ which is a `Maybe` and flattens it.
- `orElse`: Takes in $f : () \to \text{X}$, if `Some` return x, else produce the value of the producer ie $f()$
- `ifPresent`: Takes in $f : X \to \text{void}$. Only if x is present then consume the x.

Variable capture

- Things can disappear from the stack
- If a inner class uses a variable that is declared in an "outer" method

Anonymous class

- Declare a local class and instantiate in one statement
- Has the form `new X (arguments) { body }`
    - X is the class/interface that you inherit from
    - body is the methods of that class, just no constructor

Functions and λ-functions

- If an anonymous class implements an interface with one method
- Essentially a function (since there is only one method to be called)
- λ function is basically an "anonymous" function
  - Has one method so it is clear which method is overridden
- Replace these functional interface with lambda expression
  - `(variables) -> { body }
  - can omit type of variables and { } if it is a single return statement
- For stack and heap
  - Treat anonymous functions as anonymous classes
- There are more concepts (currying, closures) refer to notes for them

Question 1.

```java
Maybe<Internship> match(Resume r) {
  if (r == null) {
    return Maybe.none();
  }
  Maybe<List<String>> optList = r.getListOfLanguages();
  List<String> list;
  if (optList.equals(Maybe.none())) {
    list = List.of();
  } else {
    list = optList.get(); // cannot call
  }
  if (list.contains("Java")) {
    return Maybe.of(findInternship(list));
  } else {
    return Maybe.none();
  }
}
```

Q1.

- Convert the code to be a single statement
    - No additional classes or methods beyond those in the code
    - must not use `null` or `get`
    - no if-else statements/ternaries

Q2.

Draw stack and heap for the following

```
class A {
  private int x;
  public A(int x) {
    this.x = x;
  }
  public int get() {
    // Line A
    return this.x;
  }
}
// in main method
A a = new A(5);
Producer<Integer> p = () -> a.get();
p.produce();
```