CS2030S Recitation Problem Set 5

Brian

Recap

Fully qualified name

- Consider the following code

```
class A {
  int x = 0;

  int f() {
    int x = 3;
    return x; // hi i'm x, but which x?
  }
}
```

- What is `x` referring to?
- Somewhat ambiguous, esp from the perspective of our dumb compiler
- We can fully qualify the name to prevent ambiguity

Fully qualified name

- Consider the following code

```
class A {
  int x = 0;

  int f() {
    int x = 3;
    return this.x; // If i want to refer to field
  }
}
```

- What is `x` referring to?
- Somewhat ambiguous, esp from the perspective of our dumb compiler
- We can fully qualify the name to prevent ambiguity

Fully qualified name

- Idea is to remove ambiguity
- If it's a field add `this`
- If it's some outer class add the class name e.g. `B`
- We can chain these 2, e.g. `B.this` to access outer class B's fields

Variable capture

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

Immutability

- Slowly we are setting the stage for another paradigm
- Has to data structures or objects. NOT variables
- saying a variable/field is FINAL just means no reassginments
- Mutability has to do with whether a DS/object can mutate
- In this course we want immutable objects to have no observable change on the *outside*

Why make things immutable

- Easier to reason about
    - Guarantees that whatever you are referring to has not changed
- Sharing objects
    - Multiple objects can refer to something without worry
- Sharing internals
    - Similar to prev but we can reuse some internals (see notes example on `ImmutableSeq`)
- Safer concurrency
    - Guarantees would still hold even if different interleaving of instructions (not impt now learn next time)

Steps

- Make fields final
- Make class final
- Any mutating methods (usually return void) should now return a new instance of that class (if modifying)

The end