Tutorial 5 - Mutability & Debugging

01 More on Mutability (Achieving Immutability in Lab 4)

Consider test from DNA lab:

```
public void test_7_mutate() {
    DNA dna7 = new DNA("AAAGGTTACTG+A");
    dna.7.mutateCodon("TGA", "GAT");
    assertEquals("AAAGGTTACGAT", dna7.sequence());
}
```

Immutability:

Old definition: When a type is immutable, it means the *internal state ie. fields* of its objects cannot change after construction.

New definition: Property of a data type that characterizes whether or not *the abstract value of its objects* cannot change can never change after construction.

- Second definition is broader (technically weaker)
- In second definition, **fields** of an object can be mutable, but the abstract values of the object cannot change

To make a type immutable, we need to make sure it cannot be mutated.

- Find all ways that the abstract value of the class objects may change after construction, and restrict it.
- General practice:
  - Make all fields final
  - Return copies of abstract values if they are immutable

ex. Make ProbableGroup immutable from Lab 4.

```
public class ProbableGroup {
    // Make all fields final so they cannot be reassigned.
    private final Set<String> elements;
    private final Map<Pair<String, String> opt;
    private final String identity;

    public ProbableGroup(Set<String> elements, Map<Pair<String>, String> opTable) {
        // Create deep copy in constructor
        this.elements = Set.copyOf(new HashSet<>(elements));

        Map<Pair<String>, String> clone = new HashMap<>();
        this.opt = new HashMap<>();
        for (Map.Entry<Pair<String>, String> entry : opTable.entrySet()) {
            Pair<String> p = entry.getKey();
            Pair<String> pCopy = new Pair<>(p.getElem1(), p.getElem2());
            clone.put(pCopy, entry.getValue());
        }
        this.opt = Map.copyOf(clone);
        this.identity = this.findIdentity();

    }
}
```

02 Systematic Debugging

03 Lab 5 ADTs (The JobManager): Getting Started