Lecture – 9

Immutable Classes and
String as Immutable Class

Mutable Classes

- **mutation**: A modification to the state of an object.
- Objects can be mutable or immutable, depending on whether they can be changed after they are created.
- Mutable classes are classes whose instances can be changed after they are created.
  - Examples of mutable classes in Java include ArrayList, StringBuilder, and HashMap.

Immutable Classes

- **immutable**: Unable to be changed (mutated).
  - Basic idea: A class with no "set" methods (*mutators*).
- In Java, Strings are immutable.
  - Many methods appear to "modify" a string.
  - But actually, they create and return a new string (*producers*).

"Modifying" strings

- What is the output of this code?

```
String name = "pakistan";
name.toUpperCase();
System.out.println(name);
```

- The code outputs `pakistan` in lowercase.
- To capitalize it, we must reassign the string:

```
name = name.toUpperCase();
```

  – The `toUpperCase` method is a producer, not a mutator.

If Strings were mutable...

- What could go wrong if strings were mutable?

```
public Employee(String name, ...) {
    this.name = name;
    ...
}

public String getName() {
    return name;
}
```

  - A client could accidentally damage the Employee's name.

```
String s = myEmployee.getName();
s.substring(0, s.indexOf(" "));  // first name
s.toUpperCase();
```

Making a class immutable

- Don't provide any methods that modify the object's state.
- Declare the class as final (class cannot be extended)(later)
- Make all fields `final`.
- Make all fields `private`. (ensure encapsulation)
- Ensure exclusive access to any mutable object fields.
  - Don't let a client get a reference to a field that is a mutable object(Example Student dateCreated).

Mutable Fraction class

```
public class Fraction {
    private int numerator, denominator;

    public Fraction(int n)
    public Fraction(int n, int d)
    public int getNumerator(),  getDenominator()

   public void add(Fraction other) {
     numerator = numerator * other.denominator
               + other.numerator * denominator;
     denominator = denominator * other.denominator;
}

}
```

Immutable methods

```
// mutable version
public void add(Fraction other) {
    numerator = numerator * other.denominator
              + other.numerator * denominator;
    denominator = denominator * other.denominator;
}

// immutable version
public Fraction add(Fraction other) {
    int n = numerator  * other.denominator
          + other.numerator * denominator;
    int d = denominator * other.denominator;
    return new Fraction(n, d);
}
```

- former mutators become *producers*
  - create/return a new immutable object rather than modifying this one