CSC319 – J3 Refactoring Patterns Using Lambda Expressions

Due date: 23 May 2024, by NOON on CSCMS
Instruction: Work the following problems and zip all your answers into one (1) single *.zip* file for submission.

For each of the given code in each question, refactor using lambda expressions or method references before implementing it to verify the correct functionality.

Q1. (What pattern is this?)

```
public interface ValidationStrategy {
    boolean execute(String s);
}


public class IsAllLowerCase implements ValidationStrategy {
    public boolean execute(String s){
        return s.matches("[a-z]+");
    }
}

public class IsNumeric implements ValidationStrategy {
    public boolean execute(String s){
        return s.matches("\\d+");
    }
}


public class Validator {
    private final ValidationStrategy strategy;
    public Validator(ValidationStrategy v) {
        this.strategy = v;
    }
    public boolean validate(String s) {
        return strategy.execute(s);
    }
}
```

// Running the program

```
Validator numericValidator = new Validator(new IsNumeric());
boolean b1 = numericValidator.validate("aaaa");
Validator lowerCaseValidator = new Validator(new IsAllLowerCase ());
boolean b2 = lowerCaseValidator.validate("bbbb");
```

Returns false

Returns true

Q2. (What pattern is this?)

```
interface Observer {
    void notify(String tweet);
}

interface Subject {
    void registerObserver(Observer o);
    void notifyObservers(String tweet);
}

class NYTimes implements Observer {
    public void notify(String tweet) {
        if(tweet != null && tweet.contains("money")){
            System.out.println("Breaking news in NY! " + tweet);
        }
    }
}
class Guardian implements Observer {
    public void notify(String tweet) {
        if(tweet != null && tweet.contains("queen")){
            System.out.println("Yet more news from London... " + tweet);
        }
    }
}
class LeMonde implements Observer {
    public void notify(String tweet) {
        if(tweet != null && tweet.contains("wine")){
            System.out.println("Today cheese, wine and news! " + tweet);
        }
    }
}

class Feed implements Subject {
    private final List<Observer> observers = new ArrayList<>();
    public void registerObserver(Observer o) {
        this.observers.add(o);
    }
    public void notifyObservers(String tweet) {
        observers.forEach(o -> o.notify(tweet));
    }
}
```

// Running the program

```
Feed f = new Feed();
f.registerObserver(new NYTimes());
f.registerObserver(new Guardian());
f.registerObserver(new LeMonde());
f.notifyObservers("The queen said her favourite book is Modern Java in Action!");
```

Q3. (What pattern is this?)

```
public class ProductFactory {
    public static Product createProduct(String name) {
        switch(name){
              case "loan": return new Loan();
              case "stock": return new Stock();
              case "bond": return new Bond();
              default: throw new RuntimeException("No such product " + name);
          }
      }
  }
```

// Running the program

```
Product p = ProductFactory.createProduct("loan");
```

// Hint: Consider using the code below for refactoring

```
final static Map<String, Supplier<Product>> map = new HashMap<>();
static {
    map.put("loan", Loan::new);
    map.put("stock", Stock::new);
    map.put("bond", Bond::new);
}
```

Q4. (Short Comment) What is your opinion on refactoring design patterns using the lambdas?