```
public class RangeCombiner {
  private List<Double> mins = new ArrayList<>();
  private List<Double> maxs = new ArrayList<>();

  public void addRange(double min, double max) {
    if (max < min) { return; }
    if (mins.isEmpty() || max < mins.get(0)) { // If it should go in the front
      mins.add(0, min); maxs.add(0, max);
      return;
    }
    for (int i = 0; i < mins.size(); i++) {
      if (rangesOverlap(mins.get(i), maxs.get(i), min, max)) {
        // Set combined range at current location
        double newMin = Math.min(mins.get(i), min);
        double newMax = Math.max(maxs.get(i), max);
        insertValueAtIndexAndFixForward(newMin, newMax, i);
        return;
      }
    }
    // Did not end up merging, new range goes to end
    mins.add(min); maxs.add(max);
  }

  private boolean rangesOverlap(double min1, double max1, double min2, double max2) {
    return (max1 >= min2 && min1 <= min2) || (max2 >= min1 && min2 <= min1);
  }

  private void insertValueAtIndexAndFixForward(double currMin, double currMax, int i) {
    mins.set(i, currMin); maxs.set(i, currMax);
    // Need to possibly merge it with followup ranges
    // As long as i is not the last index:
    while (i + 1 < mins.size()) {
      Double nextMin = mins.get(i + 1);
      Double nextMax = maxs.get(i + 1);
      if (rangesOverlap(currMin, currMax, nextMin, nextMax)) {
        currMin = Math.min(currMin, nextMin);
        currMax = Math.max(currMax, nextMax);
        mins.set(i, currMin); maxs.set(i, currMax);
        mins.remove(i + 1); maxs.remove(i + 1);
      }
      break;
    }
  }

  boolean isRangeOrderValid() {
    for (int i = 0; i < mins.size() - 1; i++) {
      if (maxs.get(i) >= mins.get(i + 1)) { return false; }
    }
    return true;
  }

  private void printRanges() {
    for (int i = 0; i < mins.size(); i++) {
      System.out.println(String.format("%.2f--%.2f", mins.get(i), maxs.get(i)));
    }
  }

  public static void main(String[] args) {
    RangeCombiner combiner = new RangeCombiner();
```

```
    combiner.addRange(2.4, 3.7);
    combiner.addRange(5.6, 5.7);
    combiner.addRange(3.5, 3.8);
    combiner.addRange(6.3, 5.7); // empty range, should ignore
    combiner.addRange(5.7, 5.9);
    if (!combiner.isRangeOrderValid()) {
      System.out.println("Invalid order!");
    }
    combiner.printRanges(); // Should print 2.40--3.80 and 5.60--5.90
  }
}
```

Transformed:

```
import java.util.ArrayList;
import java.util.List;

public class RangeCombiner {
  private List<Range> ranges = new ArrayList<>();

  public void addRange(double min, double max) {
    addRangeInternal(new Range(min, max));
  }

  private void addRangeInternal(Range range) {
    if (range.isEmpty()) { return; }
    // If it should go in the front
    if (ranges.isEmpty() || range.precedes(ranges.get(0))) {
      ranges.add(0, range);
      return;
    }
    for (int i = 0; i < ranges.size(); i++) {
      Range currRange = ranges.get(i);
      if (currRange.overlapsWith(range)) {
        insertValueAtIndexAndFixForward(currRange.mergedWith(range), i);
        return;
      }
    }
    // Did not end up merging, new range goes to end
    ranges.add(range);
  }

  private void insertValueAtIndexAndFixForward(Range currRange, int i) {
    ranges.set(i, currRange);
    // Need to possibly merge it with followup ranges
    // As long as i is not the last index:
    while (i + 1 < ranges.size()) {
      Range nextRange = ranges.get(i + 1);
      if (currRange.overlapsWith(nextRange)) {
        currRange = currRange.mergedWith(nextRange);
        ranges.set(i, currRange);
        ranges.remove(i + 1);
      }
      break;
    }
  }

  boolean isRangeOrderValid() {
    for (int i = 0; i < ranges.size() - 1; i++) {
```

```
      if (!ranges.get(i).precedes(ranges.get(i+1))) {
        return false;
      }
    }
    return true;
  }

  private void printRanges() {
    for (Range range : ranges) {
      System.out.println(range.format());
    }
  }

  public static void main(String[] args) {
    RangeCombiner combiner = new RangeCombiner();
    combiner.addRange(2.4, 3.7);
    combiner.addRange(5.6, 5.7);
    combiner.addRange(3.5, 3.8);
    combiner.addRange(6.3, 5.7); // empty range, should ignore
    combiner.addRange(5.7, 5.9);
    combiner.addRange(1.1, 1.4); // should appear first
    if (!combiner.isRangeOrderValid()) {
      System.out.println("Invalid order!");
    }
    combiner.printRanges();
    // Should print 1.10--1.40, 2.40--3.80 and 5.60--5.90
  }

  private static class Range {
    private final double min;
    private final double max;

    private Range(double min, double max) {
      this.min = min;
      this.max = max;
    }

    private boolean isEmpty() {
      return max < min;
    }

    private boolean overlapsWith(Range range) {
      return (max >= range.min && min <= range.min) ||
          (range.max >= min && range.min <= min);
    }

    private Range mergedWith(Range range) {
      return new Range(Math.min(min, range.min),
                       Math.max(max, range.max));
    }

    private boolean precedes(Range range) {
      return max < range.min;
    }

    private String format() {
      return String.format("%.2f--%.2f", min, max);
    }
  }
```

```
}
```