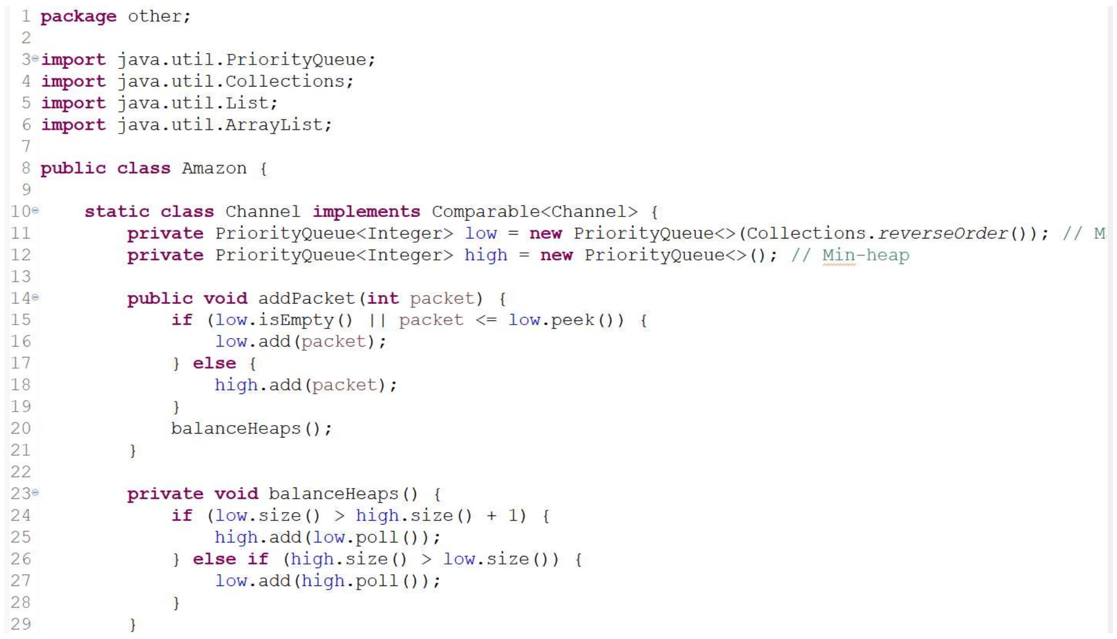
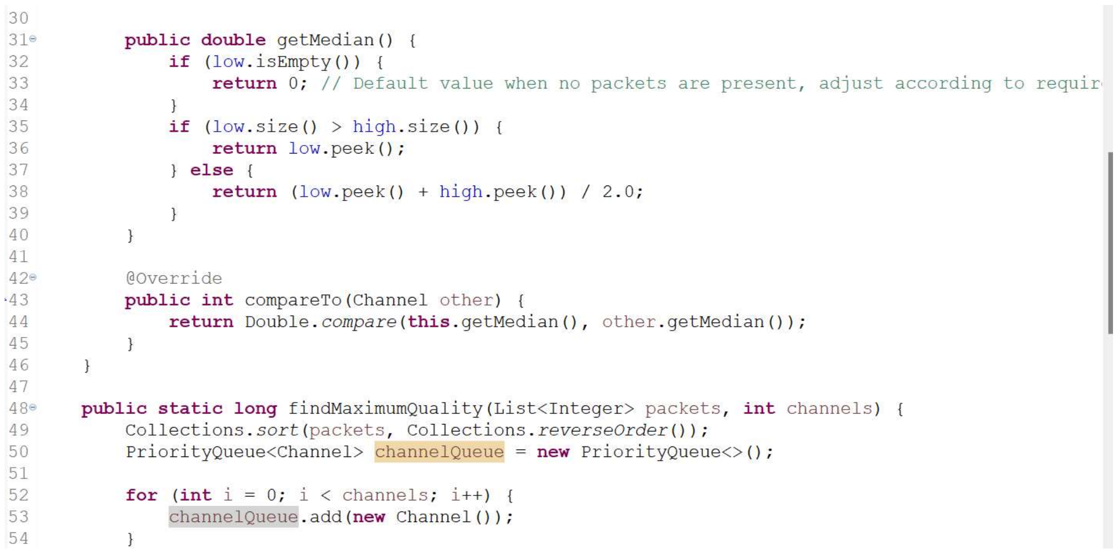
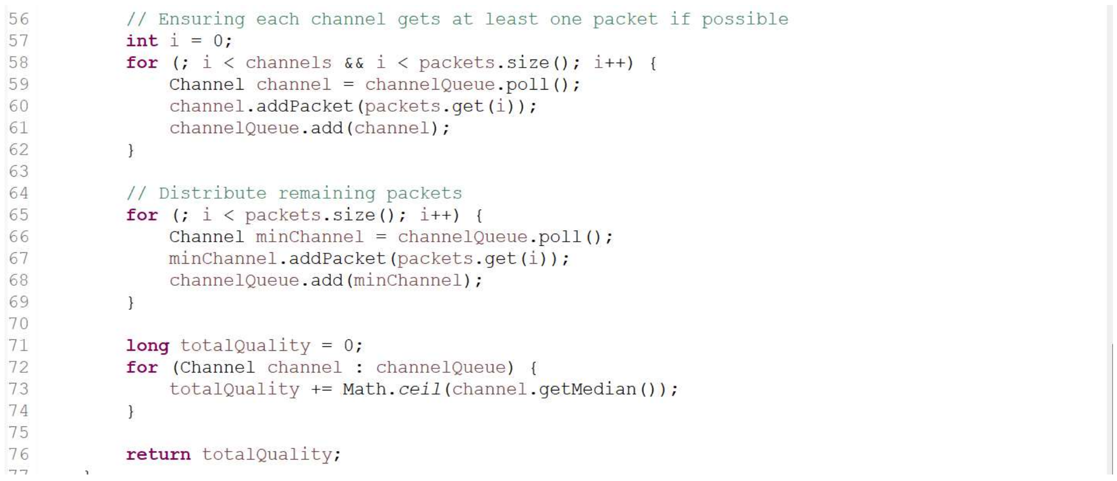


```java
package other;

import java.util.PriorityQueue;
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;

public class Amazon {

    static class Channel implements Comparable<Channel> {
        private PriorityQueue<Integer> low = new PriorityQueue<>(Collections.reverseOrder()); // M
        private PriorityQueue<Integer> high = new PriorityQueue<>(); // Min-heap

        public void addPacket(int packet) {
            if (low.isEmpty() || packet <= low.peek()) {
                low.add(packet);
            } else {
                high.add(packet);
            }
            balanceHeaps();
        }

        private void balanceHeaps() {
            if (low.size() > high.size() + 1) {
                high.add(low.poll());
            } else if (high.size() > low.size()) {
                low.add(high.poll());
            }
        }
```



```java

        public double getMedian() {
            if (low.isEmpty()) {
                return 0; // Default value when no packets are present, adjust according to requir
            }
            if (low.size() > high.size()) {
                return low.peek();
            } else {
                return (low.peek() + high.peek()) / 2.0;
            }
        }

        @Override
        public int compareTo(Channel other) {
            return Double.compare(this.getMedian(), other.getMedian());
        }
    }

    public static long findMaximumQuality(List<Integer> packets, int channels) {
        Collections.sort(packets, Collections.reverseOrder());
        PriorityQueue<Channel> channelQueue = new PriorityQueue<>();

        for (int i = 0; i < channels; i++) {
            channelQueue.add(new Channel());
        }
```



```java
        // Ensuring each channel gets at least one packet if possible
        int i = 0;
        for (; i < channels && i < packets.size(); i++) {
            Channel channel = channelQueue.poll();
            channel.addPacket(packets.get(i));
            channelQueue.add(channel);
        }

        // Distribute remaining packets
        for (; i < packets.size(); i++) {
            Channel minChannel = channelQueue.poll();
            minChannel.addPacket(packets.get(i));
            channelQueue.add(minChannel);
        }

        long totalQuality = 0;
        for (Channel channel : channelQueue) {
            totalQuality += Math.ceil(channel.getMedian());
        }

        return totalQuality;
    }
```