Java Technical Test

Introduction

A *limit order book* stores customer orders on a *price time priority* basis. The highest bid and lowest offer are considered "best" with all other orders stacked in price levels behind. In this test, the best order is considered to be at level 1.

Part A

Given the following *Order* class create an *OrderBook* class to provide efficient *limit order book* capabilities.

```
public class Order {
  private long id; // id of order
  private double price;
  private char side; // B "Bid" or O "Offer"
  private long size;

  public Order(long id, double price, char side, long size) {
    this.id=id;
    this.price=price;
    this.size=size;
    this.side=side;
  }

  public long getId() { return id; }
  public double getPrice(){ return price; }
  public long getSize(){ return size; }
  public char getSide() { return side; }
}
```

Your *OrderBook* class should support the following use-cases:

- Given an *Order*, add it to the *OrderBook* (order additions are expected to occur extremely frequently)
- Given an *order id*, remove an *Order* from the *OrderBook* (order deletions are expected to occur at approximately 60% of the rate of order additions)
- Given an *order id* and a new *size*, modify an existing order in the book to use the new size (size modifications do not effect time priority)
- Given a *side* and a *level* (an integer value >0) return the price for that level (where level 1 represents the best price for a given side). For example, given side=B and level=2 return the second best bid price
- Given a *side* and a *level* return the total size available for that level
- Given a *side* return all the orders from that side of the book, in level- and time-order

Part B

Please suggest (but do not implement) modifications or additions to the *Order* and/or *OrderBook* classes to make them better suited to support real-life, latency-sensitive trading operations.