```
import java.util.Arrays;
import java.util.Comparator;
import java.util.PriorityQueue;

class Item {
	float weight;
	int value;

	Item(float weight, int value) {
		this.weight = weight;
		this.value = value;
	}
}

class Node {
	int level, profit, bound;
	float weight;

	Node(int level, int profit, float weight) {
		this.level = level;
		this.profit = profit;
		this.weight = weight;
	}
}

public class KnapsackBranchAndBound {
	static Comparator<Item> itemComparator = (a, b) -> {
		double ratio1 = (double) a.value / a.weight;
		double ratio2 = (double) b.value / b.weight;
		// Sorting in decreasing order of value per unit weight
		return Double.compare(ratio2, ratio1);
	};

	static int bound(Node u, int n, int W, Item[] arr) {
		if (u.weight >= W)
			return 0;

		int profitBound = u.profit;
		int j = u.level + 1;
		float totalWeight = u.weight;

		while (j < n && totalWeight + arr[j].weight <= W) {
			totalWeight += arr[j].weight;
```

```
                profitBound += arr[j].value;
                j++;
            }

            if (j < n)
                profitBound += (int) ((W - totalWeight) * arr[j].value /
arr[j].weight);

            return profitBound;
        }

        static int knapsack(int W, Item[] arr, int n) {
            Arrays.sort(arr, itemComparator);
            PriorityQueue<Node> priorityQueue =
            new PriorityQueue<>((a, b) -> Integer.compare(b.bound,
a.bound));
            Node u, v;

            u = new Node(-1, 0, 0);
            priorityQueue.offer(u);

            int maxProfit = 0;

            while (!priorityQueue.isEmpty()) {
                u = priorityQueue.poll();

                if (u.level == -1)
                    v = new Node(0, 0, 0);
                else if (u.level == n - 1)
                    continue;
                else
                    v = new Node(u.level + 1, u.profit, u.weight);

                v.weight += arr[v.level].weight;
                v.profit += arr[v.level].value;

                if (v.weight <= W && v.profit > maxProfit)
                    maxProfit = v.profit;

                v.bound = bound(v, n, W, arr);

                if (v.bound > maxProfit)
                    priorityQueue.offer(v);
                v = new Node(u.level + 1, u.profit, u.weight);
                v.bound = bound(v, n, W, arr);

                if (v.bound > maxProfit)
                    priorityQueue.offer(v);
            }

            return maxProfit;
        }

        public static void main(String[] args) {
            int W = 10;
            Item[] arr = {
                new Item(2, 40),
                new Item(3.14f, 50),
                new Item(1.98f, 100),
                new Item(5, 95),
                new Item(3, 30)
            };
            int n = arr.length;

            int maxProfit = knapsack(W, arr, n);
            System.out.println("Maximum possible profit = " + maxProfit);
        }
    }
```

OUTPUT

```
Maximum possible profit = 235
```