Imagine you are a delivery company. You work with several supplier companies. Your clients order items from particular suppliers.

Your task is to aggregate orders and send combined orders to suppliers, and then notify clients about delivery. You can't aggregate orders from different suppliers, but can aggregate different items within single supplier. One client may order one or more item(s) from one or more supplier(s).

For example:

Let suppliers be: A, B, C, D.

Items that are available for delivery from each of them: a, b, c.

Your clients:

- client 1 orders:

                - [a,b] from A, B

- client 2 orders:

                - [a] from C

                - [b,c] from B

- client 3 orders:

                - [a] from B

                - [b,c] from D

You aggregate orders in the following way:

- combined order to A: [a,b]

- combined order to B: [a,b,c]

- combined order to C: [a]

- combined order to D: [b,c]

Use one of the following languages (or similar): TypeScript, JavaScript, C#, Java.

Implement using OOP.

Your program is intended to be a library, so you don't need any input parsing and output formatting, but if you want you can use it from other code (that can use files, console, etc.). External code should be able to set all required parameters and get result.

Assume all identifiers to be strings (clients, suppliers, items). You don't need to process quantity of items: if single item is ordered, let it be received in infinite quantity, enough to give it to all clients.

Your "client" object should receive notifications from your company that his (and only his) order is delivered (delivery is a pair of [supplier, item(s)]). Don't call it, just implement (you may create tests for it if you want, but it's not obligatory).

Pay attention to design and naming. You may add any other entities which you need/want.