Transaction Converter Application

Problem tryng to Solve:

1. Application tries to solve the uecase of json format transactions, but it is designed in such a way that core logic in independent of the file format and solution is easily extensible to handle other file formats

Design Patterns Used:

1. For the core logic I have used **Chain Of Responsibility(COR)** pattern.
   Core logic works on Standaridized transaction formate and is **abstracted** from the file format, so this can be reused for other formats.
   COR is used so that any new business logic on the transaction can be easily applied just adding on more chain to flow and no change to the existing classes following **Open/Closed** principle
2. **Strategy Pattern** is used for File Parser, and a **factory** is used to fetch the respective file parser based on format.
3. **Strategy Pattern** is used for **StandardizedCOnverter** from raw transaction and again factory will be repsonsible to provide respective converter based on the type.
4. **Orchestractor pattern** is used to **stitch** this flow together, each file format will have its own Orchestrator Implementation with a **Factory** sitting before that.

Consequences of Patterns Used:

Use of above patterns makes the system **Highly extensible**, and separating the business logic from other things make them **evolve indepently** as there is **no tight coupling** between them.