Builder Pattern

Alex X. Liu

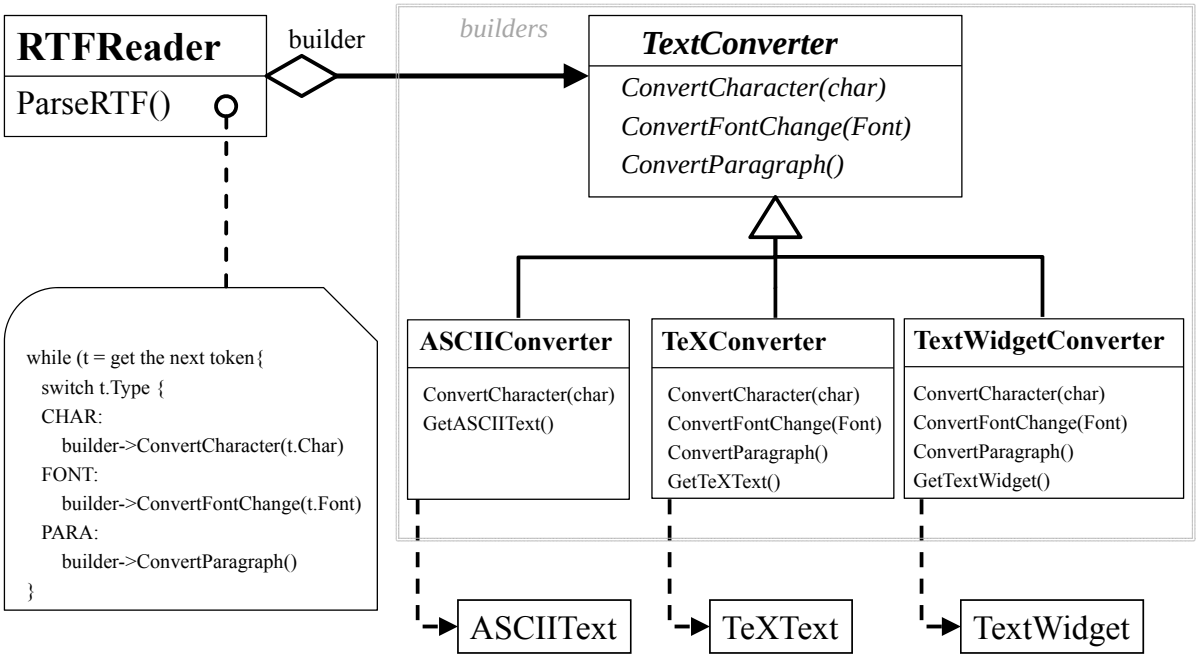
Recall: Abstract Factory Pattern

- The factory only makes parts.
- The factory does not maintain any state information.
  - I don't remember which parts you have asked me to make.
- But, sometimes it is desirable for the factory to remember state. This factory is thus called a Builder.
  - For example, a cityBuilder. It remembers what elements you have put into a city, based on which the cityBuilder design roads to connect all elements.
- The client does not need to care about the details.
  - For example, the client just call
    - cityBuilder.addPeople(10,000);
    - cityBuilder.addPostalOffice(2);
    - cityBuilder.addStore(3);
    - cityBuilder.getCity();
- Borrow ideas from the abstract factory pattern, we can have a variety of cityBuilders!

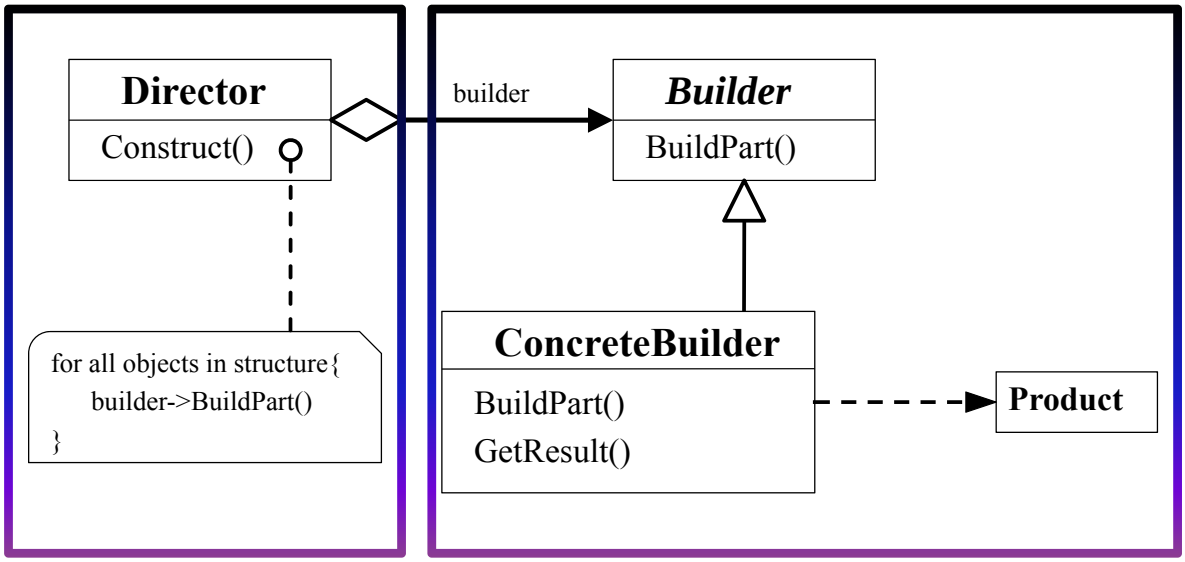
Example Application

- You want to develop a tool that can convert an RTF (Rich Text Format) document to several other formats, such as plain ASCII text, Latex text, text widget, etc.
- Design classes for this application.
- Problems:
  - Should you write a converter program for each format?
  - If you do so, all the converter programs share the same code for reading the RTF file (reading tokens, etc).
  - How to refactor all the common code into one class whose function is only reading source RTF documents?

Better Solution



Builder Pattern - UML



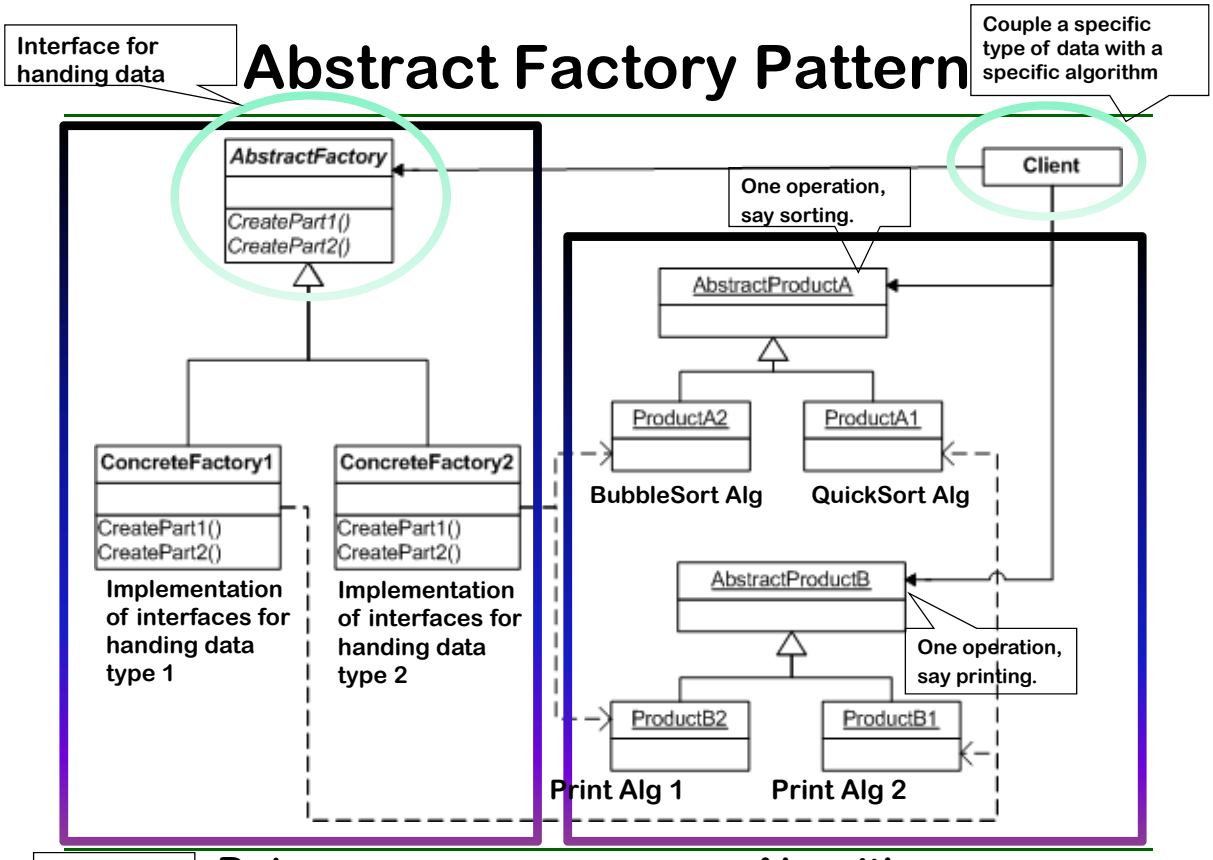
Data → Algorithms

Director **pushes** data to algorithm

Need to maintain state for storing the partial product built so far.

Algorithm class provides interfaces.

Abstract Factory Pattern



Data provides interfaces.

Data ← Algorithms

Algorithm **pulls** data

Design Virtues of Builder Pattern

- Decoupling **data preparation algorithm** and **data structure construction algorithm** (abstract side).
  - You can add new preparation algorithms without modifying the data structure construction algorithm.
  - You can add new construction algorithms (for the same data structure or for new data structures) without modifying the data preparation algorithm.
- Decoupling is generally good in software design.
  - Adding new classes or modifying classes for one side does not need to change existing classes of the other side.
- Abstract factory pattern: decouple **data** (abstract side) and **algorithm** that operate on the data.
- Whomever is called, who needs to provide abstract interfaces.
- Applicability:
  - Abstract Factory Pattern: When you want your algorithm to work for different data types.
  - Builder Pattern: When you want to build complex data structures from some data.