Builder

A creational pattern

Learning goals

1. Learn the idea, structure, and Java implementation of the Builder design pattern.
2. Learn to apply the Builder DP in your own programming.

Idea of Builder

- The Builder design pattern makes it easier to build a complex object step by step.
  - It eliminates the need for a complex, “telescoping” constructor.
- The DP separates the construction of a complex object from its representation.
  - Representation means how the object looks, how it functions and what interfaces it implements.
- It allows the same construction process to create different representations.
- Enhances code modularity and flexibility.

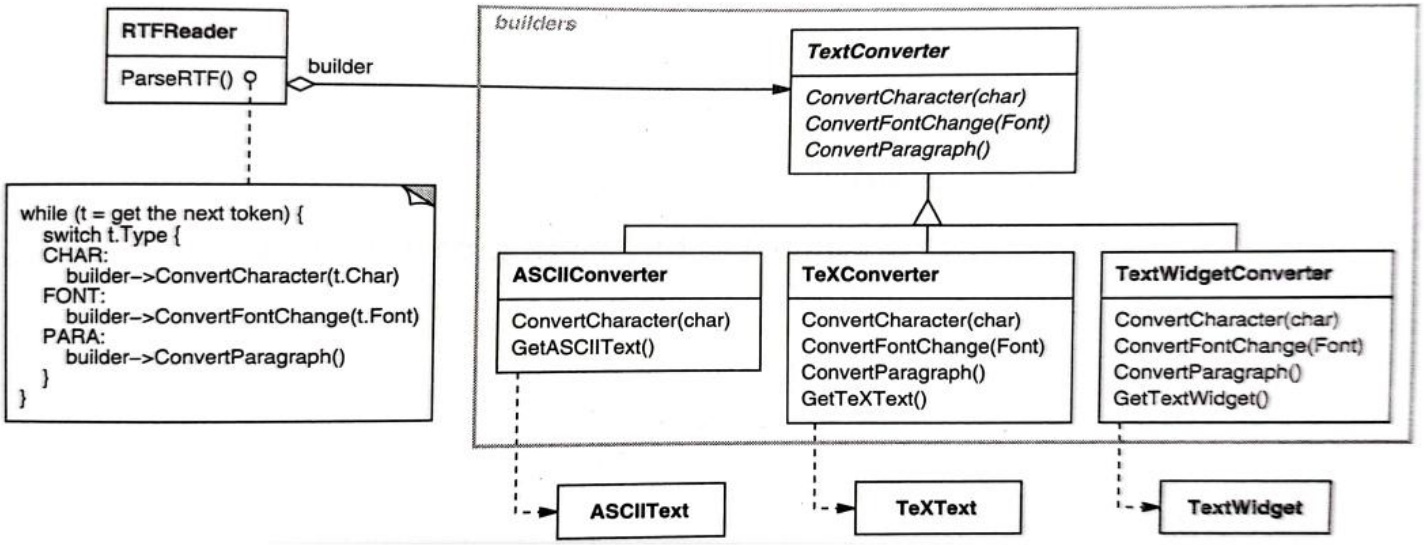
Burger Builder

Representation 1

Representation 2

Images: DALL-E v3

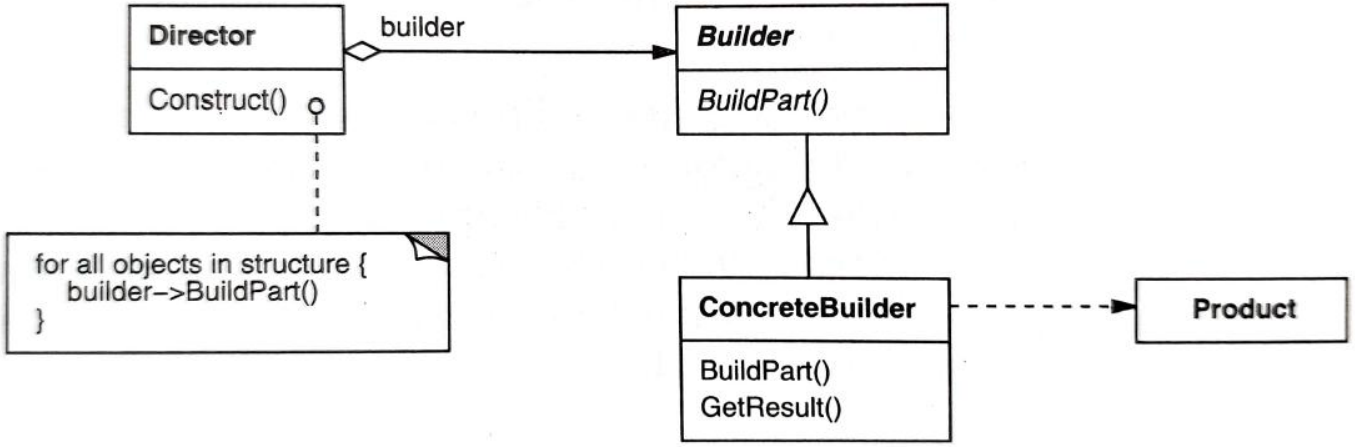
Example: Document converter



- An RTF document can be converted into an Ascii text, TeX text or into a Text Widget.
- The construction of a converted document object is complex.
  - It is infeasible to provide an overtly complex constructor for, e.g., an AsciiText object.
  - Instead, a builder object constructs an AsciiText object.

Image: Gamma et al., Design Patterns. Elements of Reusable Object-Oriented Software. Addison Wesley Longman (1995), p. 208

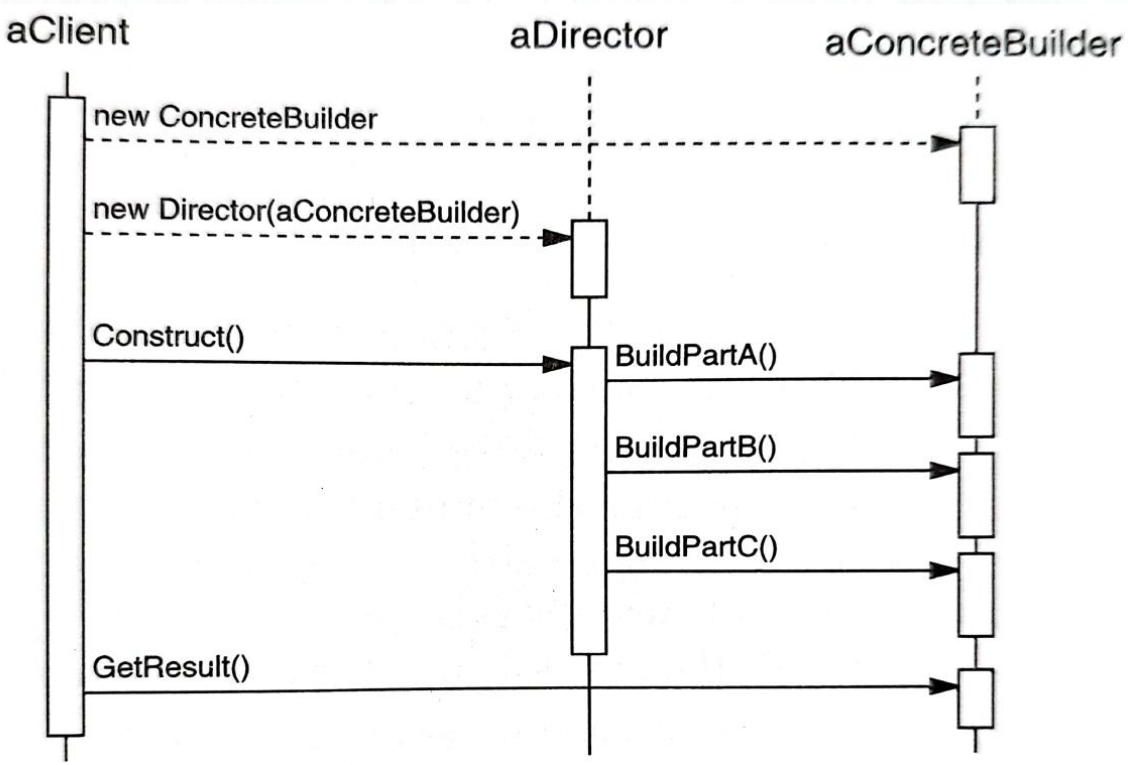
General structure



Roles

- **Builder Interface**: Defines the steps to construct the product.
- **Concrete Builder**: Implements the builder interface to construct and assemble parts of the product.
- **Director**: Orchestrates the construction with a builder instance.
- **Product**: The complex object being built.

Construction of objects



- The sequence diagram displays the process of building an object.

Getting the result

- Note that the Client fetches the result directly from the Concrete Builder.
- This approach provides the Client with the specific type of product that the Concrete Builder is responsible for creating.
- Any Director can work with any ConcreteBuilder. The Products need not share a common interface.

Example: java.lang.StringBuilder

- **Practical application**: The StringBuilder class of Java API is a practical implementation of the Builder pattern.
- **Step-by-step construction**: It provides an efficient way of constructing a complex object (String) by providing chainable building methods.
- **Encapsulation of complexity**: It encapsulates the complexity of managing a mutable sequence of characters.
- **Retrieving the product**: After the construction process, StringBuilder allows the retrieval of the final String product using the toString() method, which converts the built sequence of characters into a consolidated String.

```
StringBuilder sb = new StringBuilder();

sb.append("Hello")
      .append(" ")
      .append("World")
      .append("!")
      .insert(5, ",")
      .replace(7, 12, "Java");

String finalString = sb.toString();
System.out.println(finalString);
```

Practical issues

- The Builder pattern can introduce unnecessary complexity if the project does not require dynamic object creation.
  - For simple objects, using a Builder might overcomplicate the design.
  - Builders can increase the number of classes and the overall size of the codebase, as each new type of product requires a new concrete builder class.
- The Builder pattern may obscure the final product representation from the client.
- The readability of the client code can suffer.
  - A sequence of builder method calls can be less clear than setting properties directly on an object.