1. The **Builder Design Pattern** is a *creational* pattern used to construct complex objects step by step. Unlike constructors that can become messy and inflexible when an object has many optional parameters, the Builder pattern provides a clean and readable way to construct such objects.  
     
     
     
   This becomes unreadable and hard to maintain as the number of parameters grows, especially when they are optional or when object construction requires logic. And its solution is builder pattern.
2. Its structure:  
     
   **Product**: The complex object to be built (Report)

**Builder**: Abstract interface to build parts of the Product

**ConcreteBuilder**: Implements the Builder interface

**Director** *(optional)*: Uses builder to construct the object step-by-step

**Client**: Initiates the building  
  
  
  
  
A screenshot of a computer program

AI-generated content may be incorrect.

1. Fluent Interface Alternative (More Modern)  
     
   A screenshot of a computer program

   AI-generated content may be incorrect.  
     
   A computer code on a white background

   AI-generated content may be incorrect.
2. Why is the builder pattern needed:  
     
   a. **Too Many Constructor Parameters:** for example -   
   A close-up of a white background

   AI-generated content may be incorrect.  
     
   b. **Object Construction Logic Can Be Complex:** If building an object requires Validations, Dependencies, Conditional Steps. Then putting all that in a constructor makes the code violate SRP and OCP principles.
3. The Builder Pattern **separates the construction logic** from the object's representation.
4. The Builder pattern is the foundation of:  
   a. **LINQ query building** in C#  
   b. **HttpClient** configuration  
   c. **Entity Framework** model building  
   d. Fluent validation libraries  
   e. Fluent UI construction like HTML DSLs or UI component trees
5. Relationship between builder and factory pattern:  
   **Builder** knows *how* to assemble a complex object step by step.  
   **Factory** knows *how* to create specific parts used in that object.
6. **When to Use Builder Pattern:**a. object can be built in **many dynamic combinations.**b. Construction logic includes **validation, dependency injection, or conditionals**c. You want to let clients **choose steps dynamically.**d. You want to **compose features** without subclass explosion.  
   e. You need to **keep responsibilities separate** (like chart creation, watermarking, etc.)