Builder Design Pattern:

**What it does**

This is a creational design pattern. It creates complex and configurable object which has many parts (for example computer). It builds the object in step by step creation. In each step a mediatory object has been created.

**Where to use**

1. When you have an object with many optional components or configurations and you want to provide a clear separation between the construction process and the actual representation of the object.   
2. When the construction of an object involves a step-by-step process where different configurations or options need to be set at different stages.  
3. When the number of parameters in a constructor becomes too large, and using telescoping constructors (constructors with multiple parameters) becomes unwieldy and error-prone.  
4. When you need to create objects with different configurations or variations, and you want a more flexible and readable way to specify these configurations.  
5. When there can be some optional parameters for creating an object.  
6. When you need to create different variants of objects with similar properties but different configurations, such as different types of database connections or UI components.  
7. When you want to enforce a consistent construction process for objects across different parts of the codebase, ensuring uniformity and maintainability.

**Steps**

1. Product class: It will contain all the parts or params of a complex object. It has separate setter method for each part. Where it sets the getting value  
2. Builder abstract class: It has product object. It has builder method for each parts of the complex object.  
3. Builder concrete object: It extends the builder abstract class and implements all the builder method defined in the builder abstract class. There is called setter method of that particular part of the product which builder functions have been implemented.  
4. Director: It has the builder object and it just called the builder method of different parts of the product in a manner what is expected.  
5. Client code: It creates builder objects using concrete builder class. Then create director object using director class and pass product as parameter to its constructor.

**Advantages**

1. Without builder design pattern, we have to create too many constructors for different different permutation combination of the object parameter. Builder design pattern has solved this problem.  
2. It simplifies the creation of complex objects with multiple properties or configurations, reducing the risk of errors in object construction.  
3. Builders can create different variants of objects by adjusting construction parameters or using different builder implementations, enabling the creation of objects tailored to specific use cases or environments.  
4. The builder pattern also avoids the telescopic constructor anti-pattern and the creation of too many constructors, which can make coding and maintenance difficult.  
5. Scales well with changes in object structure or requirements, allowing for easy addition of new properties or behaviors.  
6. Enables clients to create objects without needing to know the intricate details of their construction, promoting encapsulation and abstraction.

**Disadvantages**

1. The overall complexity of the code increases since the pattern requires creating multiple new classes.  
2. Introducing a builder class and defining multiple steps for object construction can make the code more complex and harder to understand, especially for simple objects.  
3. There is a risk of creating incomplete or inconsistent objects if the builder's construction steps are not properly defined or if certain mandatory properties are not set.  
4. Developers unfamiliar with the builder pattern may find it challenging to understand and implement, leading to a steeper learning curve compared to simpler object creation methods.  
5. Added too much of redundant codes.

**Code**

Keerti Purswani

**Difference with similar pattern**

Decorator Design Pattern: While both patterns involve object manipulation and provide flexibility in object creation or modification, they serve different purposes and operate at different levels of granularity. Builder is used for constructing complex objects, while Decorator is used for dynamically extending the behavior of existing objects.

**Diagram**

Keerti Purswani