Builder-Pattern wird verwendet , um komplexe Objekte Schritt für Schritt zu erstellen. Es hilft, die Komplexität der Objektkonstruktion zu reduzieren, indem es die Erstellung in übersichtliche und logische Schritte unterteilt. **Der Builder-Pattern wird häufig verwendet, wenn ein Objekt viele Felder oder optionale Parameter hat und es schwierig ist, den Konstruktor zu lesen oder zu verwalten.**

Was ist das Ziel des Builder Patterns?

Das Ziel des **Builder-Patterns** ist es, **eine klare und flexible Möglichkeit zu bieten, ein komplexes Objekt zu erstellen**, ohne die **Reihenfolge oder Anzahl** der benötigten Parameter zu verwirren. Es wird verwendet, wenn:

- Ein Objekt viele optionale Felder hat.
- Du das Risiko von Konstruktoren mit vielen Parametern vermeiden möchtest (auch bekannt als **Telescoping Constructors Problem**).
- Du möchtest, dass dein Code lesbar bleibt und leicht zu erweitern ist.

So funktioniert der Builder-Pattern:

1. Du erstellst eine **statische innere Klasse** (den **Builder**) innerhalb der zu bauenden Klasse.
2. Der Builder hat dieselben Felder wie die Zielklasse, und für jedes Feld gibt es eine **Methode**, um den Wert festzulegen.
3. Nachdem alle Felder gesetzt wurden, erstellt der Builder das Objekt mit einer **build()**-Methode.

Vorteile des Builder-Patterns:

1. **Klare und lesbare Code-Struktur**: Statt lange Konstruktorparameter zu verwenden, kannst du Schritt für Schritt jedes Feld setzen, was den Code klar und lesbar macht.
2. **Flexibilität bei optionalen Parametern**: Du musst nicht alle Parameter angeben. Optionalen Parametern kannst du einfach keinen Wert zuweisen, und sie bleiben auf einem Standardwert.
3. **Keine mehrdeutigen Konstruktoren**: Du vermeidest das Problem von überladenen Konstruktoren mit vielen Parametern. Mit dem Builder kannst du jedes Attribut in beliebiger Reihenfolge setzen.
4. **Unveränderlichkeit des Objekts**: Das erstellte Objekt ist oft **immutable** (unveränderlich), da alle Felder `final` oder privat sind und nur einmal über den Builder gesetzt werden.
5. Keine Verwirrung bei der Reihenfolge der Parameter, die Reihenfolge der Methodenaufrufe frei bestimmen, und das ist besonders hilfreich, wenn viele Parameter vorhanden sind.

Wann sollte man den Builder-Pattern verwenden?

- **Komplexe Objekte**: Wenn ein Objekt viele Parameter hat, vor allem optionale, ist der Builder-Pattern die beste Wahl, um den Code lesbar und wartbar zu halten.
- **Unübersichtliche Konstruktoren**: Wenn du viele überladene Konstruktoren hast oder das Hinzufügen neuer Parameter schwierig ist, hilft der Builder-Pattern, diese Komplexität zu reduzieren.
- **Fluente API**: Wenn du eine "flüssige" API haben möchtest, bei der Methodenaufrufe in einer Kette zusammenhängen, ist der Builder eine ausgezeichnete Lösung.

Zusammenfassung:

Das **Builder-Pattern** ist ein sehr nützliches Entwurfsmuster, das es ermöglicht, **komplexe Objekte schrittweise zu erstellen**. Es bringt folgende Vorteile:

- **Verbesserte Lesbarkeit**: Der Code wird viel lesbarer, da er das "Telescoping Constructor"-Problem löst.
- **Flexibilität**: Es gibt dir die Flexibilität, nur die Felder zu setzen, die du tatsächlich brauchst.
- **Wartbarkeit**: Es wird einfacher, neue Felder hinzuzufügen, ohne den bestehenden Code zu ändern.
- **Pflicht- und optionale Felder**: Du kannst leicht Pflichtfelder von optionalen Feldern trennen und Standardwerte für optionale Felder festlegen.

Der Builder-Pattern wird in vielen realen Szenarien eingesetzt, insbesondere wenn du große und komplexe Objekte mit vielen Parametern bauen musst.