Folie 1.

Stellen Sie sich ein komplexes Objekt vor, das eine schrittweise Initialisierung vieler Felder und verschachtelter Objekte erfordert. Dieser Code wird normalerweise in einem riesigen Konstruktor mit vielen Parametern implementiert. Oder noch schlimmer: Es ist hier und da im gesamten Client-Code zu finden.

Wie zum Beispiel auf dieser Folie, wo man eine Vielzahl von Optionen für ein Haus mit Pool, Garten, Statuen und Garage sehen kann. Die einfachste Lösung besteht darin, die Basisklasse House zu erweitern und eine Reihe von Unterklassen zu erstellen, die alle Parameterkombinationen abdecken. Aber irgendwann wird die Anzahl der Unterklassen einige vernünftige Grenzen überschreiten. Jeder neue Parameter erfordert eine weitere Erweiterung dieser Hierarchie.

Folie 2.

Das Builder-Muster löst das Problem mit einer großen Anzahl optionaler Parameter, indem es eine Möglichkeit bietet, das Objekt schrittweise zu erstellen und eine Methode bereitzustellen, die tatsächlich das endgültige Objekt zurückgibt.

Das Builder-Entwurfsmuster ist ein Creational Pattern, das im Softwareentwicklung verwendet wird, um die komplexen Objekte schrittweise zu konstruieren. Es ermöglicht die Erstellung eines Produkts, wobei sich der Konstruktionsprozess je nach Art des zu erstellenden Produkts ändern kann. Dieses Muster trennt die Konstruktion eines komplexen Objekts von seiner Darstellung, sodass im selben Konstruktionsprozess verschiedene Darstellungen erstellt werden können.

Im Wesentlichen bedeutet er Folgendes: Das Builder-Muster ermöglicht es, dass man den gleichen Konstruktionsprozess verwenden kann, um unterschiedliche Varianten oder Darstellungen des Objekts zu erzeugen.

Folie 3.

Die Grafik zeigt, dass das Builder-Muster aus mehreren Objekten besteht, die miteinander interagieren.

## Produkt

## Das Produkt ist das komplexe Objekt, für dessen Konstruktion das Builder-Muster verantwortlich ist. Es kann aus mehreren Komponenten oder Teilen bestehen und seine Struktur kann je nach Implementierung variieren. Das Produkt ist normalerweise eine Klasse mit Attributen, die die verschiedenen, vom Builder erstellten Teile darstellen.

## Baumeister(Builder)

## Der Builder ist eine Schnittstelle oder eine abstrakte Klasse, die die Konstruktionsschritte zum Erstellen eines komplexen Objekts deklariert. Es umfasst typischerweise Methoden zum Konstruieren einzelner Produktteile. Durch die Definition einer Schnittstelle ermöglicht der Builder die Erstellung verschiedener konkreter Builder, die Variationen des Produkts erstellen können.

1. Konkrete Builder

ConcreteBuilder-Klassen implementieren die Builder-Schnittstelle und bieten spezifische Implementierungen zum Erstellen jedes Teils des Produkts. Jeder ConcreteBuilder wird individuell angepasst, um eine bestimmte Variante des Produkts zu erstellen. Es verfolgt das hergestellte Produkt und bietet Methoden zum Einstellen oder Herstellen der einzelnen Teile.

1. Direktor

Der Direktor ist für die Leitung des Bauprozesses des komplexen Objekts verantwortlich. Es arbeitet mit einem Erbauer zusammen, kennt aber nicht die spezifischen Details darüber, wie die einzelnen Teile des Objekts konstruiert sind. Es bietet eine Schnittstelle auf hoher Ebene zum Erstellen des Produkts und zum Verwalten der zum Erstellen des komplexen Objekts erforderlichen Schritte.

1. Kunde

Der Client ist der Code, der die Konstruktion des komplexen Objekts initiiert. Es erstellt ein Builder-Objekt und übergibt es an den Director, um den Konstruktionsprozess zu starten. Der Kunde kann das Endprodukt nach Abschluss der Bauarbeiten vom Bauunternehmer abholen.

Folie 4.

Was ist hier interessant? Hier können wir klarer sehen, dass der Konstruktionsprozess explizit vom Director gesteuert wird. Was als nächstes passiert (die internen Darstellungen des Builders) bleibt dem Director verborgen. Die Konstruktion und die Darstellung (Ausgabe) werden getrennt voneinander integriert. So können neue Darstellungen durch die Verwendung konkreter Konstruktionsklassen leicht integriert werden.

Folie 5

Kunden wollen unterschiedliche Handys kaufen: iPhone oder Samsung. Der Direktor sagt: „Okay, kein Problem, ich habe einen Meisteringenieur, er wird dieses Problem lösen.“ Der Meisteringenieur sagt: „In meinem Büro gibt es mehrere Mitarbeiter, von denen jeder weiß, wie man einen Telefontyp herstellt. Ich delegiere diese Aufgaben an verschiedene Mitarbeiter.

## Ein Mitarbeiter ist iPhone-Spezialist, der andere Samsung-Spezialist.

Folie 6

## **Wann sollte das Builder-Entwurfsmuster verwendet werden?**

## Das Builder-Entwurfsmuster wird verwendet, wenn Sie komplexe Objekte mit einer großen Anzahl optionaler Komponenten oder Konfigurationsparameter erstellen müssen. Dieses Muster ist besonders nützlich, wenn ein Objekt Schritt für Schritt erstellt werden muss.

## Dies ermöglicht Ihnen einen schrittweisen Aufbau und vermeidet Teleskopkonstruktoren (Konstruktoren mit mehreren Parametern),

## Konfigurierbare Objekterstellung: Wenn Sie Objekte mit unterschiedlichen Konfigurationen oder Variationen erstellen müssen und diese Konfigurationen auf flexiblere und lesbarere Weise angeben möchten.

## Gemeinsame Schnittstelle für mehrere Darstellungen: Wenn Sie eine gemeinsame Schnittstelle für die Konstruktion verschiedener Darstellungen eines Objekts bereitstellen möchten.

## **Nachteile des Builder-Musters**

## Diese Methode eignet sich nicht zum Erstellen einfacher Objekte - Wenn das von Ihnen erstellte Objekt nur wenige einfache Parameter oder Konfigurationen aufweist und der Konstruktionsprozess unkompliziert ist, ist die Verwendung eines Builders möglicherweise nicht erforderlich und kann dazu führen, dass Programmierer einfachere und elegantere Lösungen übersehen.

Folgendes hängt damit zusammen

* Erhöhte Codekomplexität - Wenn das zu erstellende Objekt einfach ist und von einem schrittweisen Konstruktionsprozess nicht wesentlich profitiert, kann die Verwendung eines Builders die Codebasis unnötig komplex machen.

## Enge Kopplung mit dem Produkt - Das Builder-Muster besteht aus einer engen Kopplung zwischen dem Produkt, dem spezifischen Builder und anderen beteiligten Klassen, sodass Änderungen am grundlegenden Prozess schwierig sein können.

Folie 7

1.Die folgenden Codebeispiele verdeutlichen die einzelnen Akteure des Builder-Patterns noch einmal. Das Objekt, also Pizza, ist zunächst leer. Inhalte werden erst nach der Bestellung hinzugefügt. Hier erstellen wir die Produktklasse. Diese Klasse enthält alle Felder, aus denen das Objekt besteht.

Folie 8

2. Der Direktor stellt die „Umgebung“ bereit, damit ein Gericht für den Gast zubereitet – also gebaut – werden kann. Der Gast kommuniziert nur mit dem Direktoren, die eigentliche Zubereitung bleibt verborgen. Direktor steuert den Erstellungsprozess und legt die Reihenfolge, in der die Teile erstellt werden, fest.

Anschließend tritt der Builder in Aktion. In unserem Beispiel wäre das ein Chef. Wir erstellen die Builder-Klasse : Diese Klasse verfügt über Methoden zum Festlegen der verschiedenen Teile des Produkts (Pizza). Jede Methode gibt den Builder selbst zurück, um eine Methodenverkettung zu ermöglichen.

Am Ende steht die Build-Method, die das Produkt zusammensetzt und das endgültige Objekt zurückgibt.

Folie 9

Der konkrete Builder, in diesem Fall der Koch, baut (also backt) die einzelnen Bestandteile des bestellten Gerichts. Dazu überschreibt er die Menüpunkte, legt die gewünschten Teile Schritt für Schritt fest und ruft die build()-Methode auf, um das Endprodukt zu erhalten.

Folie 10

1. Es wird eine Instanz von MargheritaPizzaBuilder erstellt, dem konkreten Builder für die „Margherita“-Pizza. Dieses Objekt margheritaBuilder implementiert das PizzaBuilder-Interface, welches die Schritte zur Erstellung der Pizza definiert. Nun weiß margheritaBuilder, welche Zutaten für die „Margherita“-Pizza verwendet werden (z. B. „dünner Teig“, „Tomatensoße“, „Mozzarella“).
2. Es wird ein PizzaDirector-Objekt erstellt, das den margheritaBuilder als Parameter an den Konstruktor übergibt. Der PizzaDirector steuert den Aufbauprozess der Pizza, indem er die Methoden des übergebenen Builders aufruft. Er ist dafür verantwortlich, die Schritte zur Erstellung der Pizza in der richtigen Reihenfolge auszuführen.
3. Jede Methode setzt spezifische Parameter für die „Margherita“. Am Ende gibt die Methode build() ein Pizza-Objekt mit den festgelegten Parametern zurück. Das margherita-Objekt repräsentiert nun die fertige „Margherita“-Pizza mit den gewünschten Eigenschaften.
4. Es wird ein PepperoniPizzaBuilder-Objekt erstellt, das für die Erstellung der „Pepperoni“-Pizza zuständig ist. pepperoniBuilder implementiert ebenfalls das PizzaBuilder-Interface, setzt aber andere Zutaten für die „Pepperoni“-Pizza.
5. Ein neuer PizzaDirector, der den Erstellungsprozess der Pizza steuert, verwendet diesmal pepperoniBuilder. pepperoniDirector weiß, wie man die „Pepperoni“-Pizza korrekt zusammenstellt und startet den Erstellungsprozess mit den richtigen Parametern.
6. Die Methode createPizza() im pepperoniDirector startet eine Kette von Methodenaufrufen in pepperoniBuilder, um die „Pepperoni“-Pizza zu erstellen.
7. Das Ergebnis ist ein pepperoni-Objekt, das die fertige „Pepperoni“-Pizza darstellt und dem Gast ausgeliefert wird.

## 