# Builder

Il builder è un pattern di progettazione creazione che consente di creare oggetti complessi passo dopo passo.

Ad esempio per creare l'oggetto Home è necessario costruire quattro pareti e un pavimento, installare una porta, le finestre e costruire il tetto. Ma cosa succede se si desidera aggiungere delle funzionalità a questa casa base?

La soluzione più semplice è quella di estendere la classe Home e creare un insieme di sottoclassi che coprano tutte le combinazioni dei parametri. Ma questo porterà ad avere un numero considerevole di sottoclassi.

Esiste un altro approccio che non prevede la definizione di tutte queste sottoclassi. Infatti è possibile creare un grande costruttore nella classe Home con tutti i possibili parametri che controllano l'oggetto casa. Ma questo approccio, se da una parte elimina la definizione di tante sottoclassi, crea un altro problema.

Nella maggior parte dei casi, la maggior parte dei parametri non verranno usati, rendendo le chiamate al costruttore molto brutte con molti parametri settati a null.

La soluzione viene fornita proprio dal pattern builder, il quale suggerisce di estrarre il codice di costruzione dell'oggetto dalla sua classe e spostarlo in un'apposita classe (o più) chiamata builder.

Ciascuna classe convertitore è chiamata builder, mentre il lettore è  chiamato director.

## Applicabilità

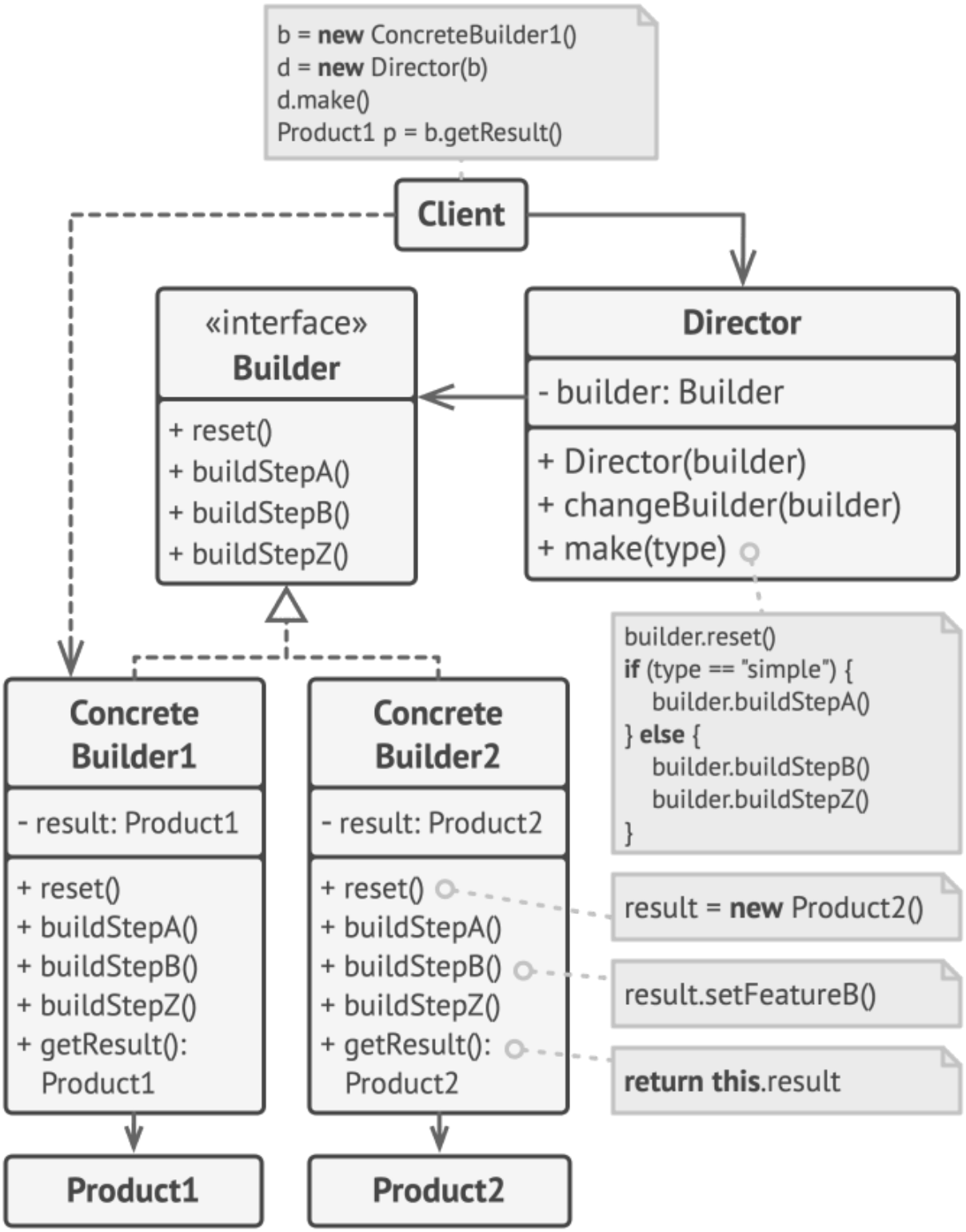
L'utilizzo del pattern Builder è consigliato quando:

- l'algoritmo per la creazione di un oggetto complesso deve essere mantenuto, separato e indipendente dalle parti di cui l'oggetto è composto e dal modo con cui queste sono assemblate;
- il processo di costruzione deve consentire rappresentazioni differenti per l'oggetto da costruire.

Inoltre:

- Tale pattern elimina il costruttore telescopico: ovvero, supponiamo di avere un costruttore con tanti parametri opzionali. Richiamare tale costruttore è molto scomodo; pertanto si sovraccarica il costruttore e si creano diverse versioni più brevi con meno parametri. Con il Builder questi passaggi non sono più necessari, infatti permette di costruire gli oggetti complessi passo dopo passo, usando solo i passaggi realmente necessari.

## Struttura




- Interfaccia Builder: Dichiara i passaggi di costruzione del prodotto comuni a tutti i tipi di builder;

- ConcreteBuilder: forniscono diverse implementazioni del builder e possono realizzare diversi prodotti che non seguono l'interfaccia comune;
- I Prodotti: sono gli oggetti risultanti. ConcreteBuilder costruisce la rappresentazione interna di Product e definisce il processo attraverso il quale il prodotto viene assemblato;
- Director: costruisce un oggetto tramite l'interfaccia Builder. Definisce l'ordine con cui richiamare i passaggi di costruzione, in modo da poter creare e riutilizzare configurazioni specifiche dei prodotti;
- Il Client: deve associare uno degli oggetti Builder al director. Essenzialmente, crea l'oggetto director e lo configura in modo da farlo operare con l'oggetto Builder desiderato.

Ogni volta che una parte di Product deve essere costruito il Director informa il Builder, il quale riceve e gestisce le richieste del Director e aggiunge le parti a Product. Infine il Client recupera dal Builder il prodotto creato.

## Caratteristiche

Le caratteristiche principale del pattern Builder sono:

- Consente di variare la rappresentazione interna di un prodotto. L'oggetto Builder fornisce al Director un'interfaccia astratta per la costruzione del Product. Tale interfaccia consente al Builder di nascondere la rappresentazione e la struttura interna del Product e il processo di costruzione del medesimo. Per modificare la rappresentazione interna del product è sufficiente definire una nuova tipologia di Builder;
- Isola il codice per la costruzione e la rappresentazione;
- Consente un migliore controllo del processo di costruzione.

## Implementazione

Tipicamente esiste una classe astratta Builder che definisce un'operazione per ciascun elemento che un Director potrebbe chiedere di creare. Queste operazioni

non fanno niente di base. Un ConcreteBuilder di base sovrascriverà le operazioni per i soli componenti che vuole creare. In particolare, avremo:

- Interfaccia per la costruzione e l'assemblaggio delle parti (Builder): I Builder costruiscono i prodotti passo dopo passo. Tale interfaccia dovrà essere generica in modo da poter costruire dei prodotti per tutti i tipi di builder concreti;
- Creare una classe di costruttori concreti per ciascuna delle rappresentazioni del prodotto e implementare le relative fasi di costruzione; (ConcreteBuilder)
- Creare la classe Director;
- Creazione e associazione.

## Vantaggi

- E' possibile creare oggetti complessi passo dopo passo, posticipare i passaggi di costruzione o eseguirli in modo ricorsivo;
- E' possibile riusare lo stesso codice di costruzione quando si creano diverse rappresentazioni di prodotti;
- Principio di singola responsabilità;

## Svantaggi

- La complessità del codice aumenta in quando si devono definire più nuove classi.

## Relazioni con altri pattern

- Molti progetti iniziano usando il Factory Method e si evolvono in AbstractFactory, Prototype o Builder;
- Builder si concentra sulla costruzione di oggetti complessi passo dopo passo e consente di eseguire alcuni passaggi di costruzione aggiuntivi prima di

recuperare l’oggetto. AbstractFactory è specializzato nella creazione di famiglie di oggetti correlati e restituisce il prodotto immediatamente;

- E’ possibile combinare Builder con Bridge: la classe Director svolge il ruolo di astrazione, mentre i diversi Builder fungono da implementazioni;
- AbstractFactory, Builder e Prototype possono essere tutti implementati come Singleton.