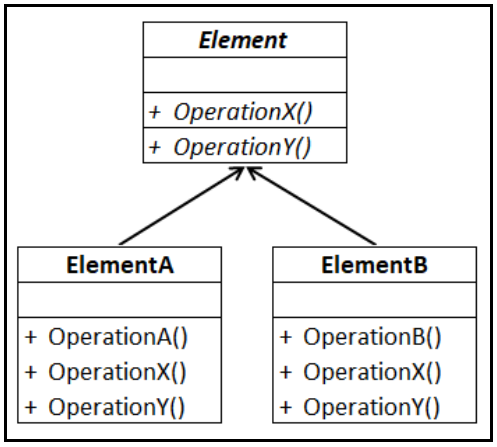
Visitor

1 Visitor

Beim Visitor-Pattern verwaltet der Client eine Struktur von Elementen (Liste, Baum, …). Auf die einzelnen Elemente dieser Struktur werden Operationen ausgeführt (z.B. Anzahl aller Elemente ermitteln, Gesamtpreis einer Einkaufsliste berechnen, …).
Diese Operationen werden jeweils in einzelnen Visitor-Klassen gekapselt. Ziel des Patterns ist es, neue Operationen hinzufügen zu können, ohne die Element-Klassen verändern zu müssen bzw. in das Abarbeiten der Objektstruktur eingreifen zu müssen.
Anstelle einer neuen Methode in allen Klassen der Ableitungshierarchie wird ein **neuer Visitor** erzeugt. Man kann damit also quasi Klassen mit Methoden erweitern, auch wenn man unter Umständen den Code dieser Klassen nicht zur Verfügung hat.
Damit das funktioniert, müssen alle Klassen eine Methode haben, der man einen derartigen Visitor übergibt.

1.1 Ohne Visitor

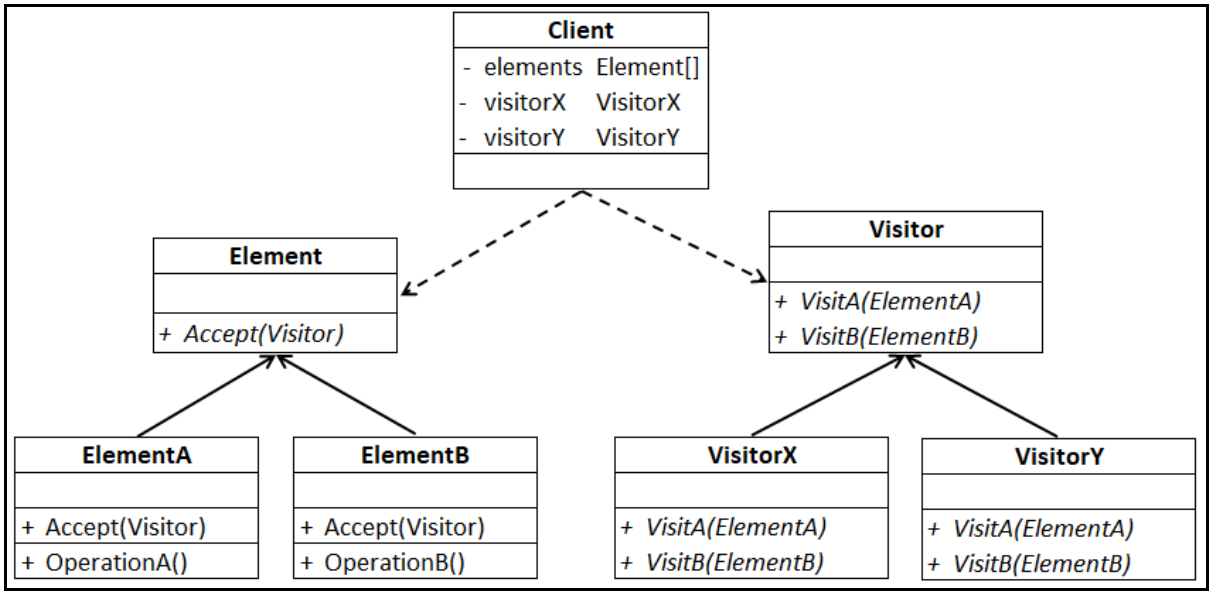
Zum besseren Verständnis soll zuerst eine kleine Klassenhierarchie ohne Visitor betrachtet werden:



Die Basisklasse definiert einige abstrakte Methoden, die in jeder abgeleiteten Klasse implementiert werden müssen. Darüber hinaus können die konkreten Klassen weitere Methoden haben.
Brauch man jetzt eine neue Methode, muss diese in der Basisklasse definiert werden und in weiterer Folge in allen Klassen programmiert werden. Das setzt also voraus, dass man den Sourcecode dieser Klassenhierarchie greifbar ist.

1.2 UML

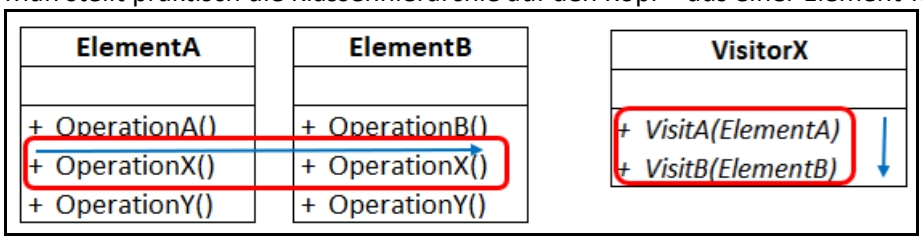
Das UML-Diagramm ist beim Visitor etwas umfangreicher, weil man zwei Hierarchien betreuen muss:



Hinweise:

- **Element**
  - Die abstrakte Basisklasse hat zumindest die abstrakte Methode Accept(Visitor).
  - Die konkreten Klassen implementieren diese Klasse dann und können entscheiden, ob sie den „Besuch“ annehmen wollen.
  - In der Methode Accept() wird dann die entsprechende Visit-Methode des Visitors aufgerufen. Dabei übergibt sich das Element selbst als Parameter.
- **Visitor**
  - Pro konkreter Element-Klasse gibt es eine Visit-Methode.
  - Die einzelnen konkreten Visitor-Klassen stehen jeweils für eine Methode in den Element-Klassen. Diese Methoden sind verschwunden. Für jede gewünschte Operation gibt es also genau einen Visitor.
  - Die einzelnen Visitor-Klassen haben oftmals zusätzliche Variable, in denen sie das Ergebnis der jeweiligen Besuche abspeichern.
- **Client**
  - Der Client hat eine Liste von Elementen.
  - Je nachdem, welche Operation er für die einzelnen Elemente aufrufen möchte, erzeugt er einen passenden Visitor, iteriert über die Elemente und übergibt sie diesem Visitor.

Man stellt praktisch die Klassenhierarchie auf den Kopf – aus einer Element-Methode wird ein neuer Visitor:



1.3 Erkennungsmerkmale

An folgenden Merkmalen kann man erkennen, dass der Einsatz eines Visitors sinnvoll ist:

- Man hat eine umfangreiche Klassenhierarchie, bei der man möglicherweise Funktionen hinzufügen möchte, ohne dass man den Sourcecode zur Verfügung hat.
- Man möchte die Schnittstellen einer Klassenhierarchie schlank halten.

1.4 Beispiele

In folgenden Beispielen würde sich ein Visitor anbieten:

- Warenkorb; die einzelnen Visitoren berechnen den Preis, das Gewicht, …
- grafische Elemente, die auf unterschiedliche Weise verändert werden, z.B. rotieren, verschieben, spiegeln, strecken, …

- Bäume mit verschiedenen Objekten, die in unterschiedliche Formate umgewandelt werden sollen.

1.5 Vor-/Nachteile

1.5.1 Vorteile

- Man erspart sich das Kompilieren einer Klassenbibliothek und kann trotzdem neue Methoden hinzufügen. Das ist vor allem dann wichtig, wenn es den Sourcecode gar nicht mehr gibt, sondern nur die DLL. In diesem Fall ist der Visitor die einzige Chance.
- Ein Visitor kapselt genau eine Aufgabe.

1.5.2 Nachteile

- Es steigt zuerst einmal die Komplexität des Projekts – es kommen ja einige Klassen hinzu.
- Man muss schon zu Beginn das Visitor-Pattern berücksichtigen (Accept-Methoden).
- Benötigt man neue konkrete Elementklassen, müssen alle Visitor-Klassen eine neue Methode implementieren.
- Der Visitor hat nur Zugriff auf die public Methoden/Properties der einzelnen Objekte.
- Das Debuggen wird eventuell etwas umständlicher.