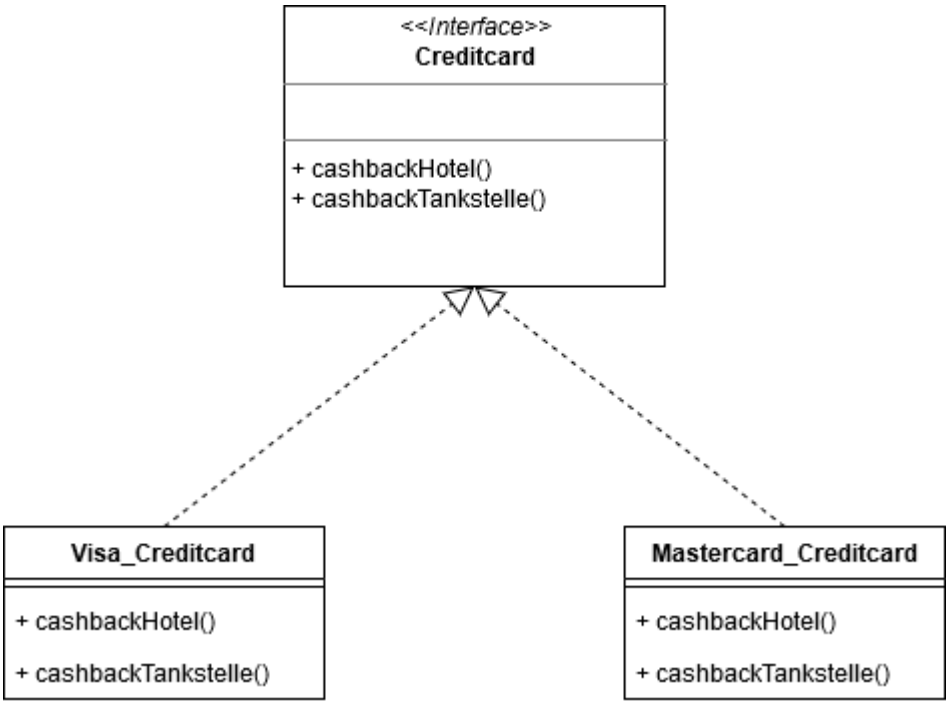
# Visitor Pattern

Ziel des Visitor Patterns ist es Daten verändern zu können ohne die eigentliche Datenklasse zu verändern. Das Visitor Pattern ist also für die Wartbarkeit und einfache Funktionserweiterung von Programmen wichtig. Ein Programm wird auch ohne Verwendung des Visitor Patterns gut funktionieren. Wie wichtig das Patterns ist zeigt sich erst später wenn unerwartete Funktionserweiterungen am Programm notwending sind.

## Worum geht es?

Nehmen wir an wir haben möchten ein System für eine Bank erstellen, welches Kreditkarten verwaltet. Die Bank hat sowohl Visa als auch Mastercard Kreditkarten. Von beiden gibt es jeweils noch verschiedene Modelle wie z.B. Classic und Gold. Die Kreditkarten nehmen an einem Cashbacksystem teil. Die Cashbackaktion unterscheidet zwischen verschiedenen Einsatzgebieten so wird das Cashback an einer Tankstelle anders berechnet als jenes eines Hotels.

## Eine typische Lösung ohne Anwendung des Visitor Patterns



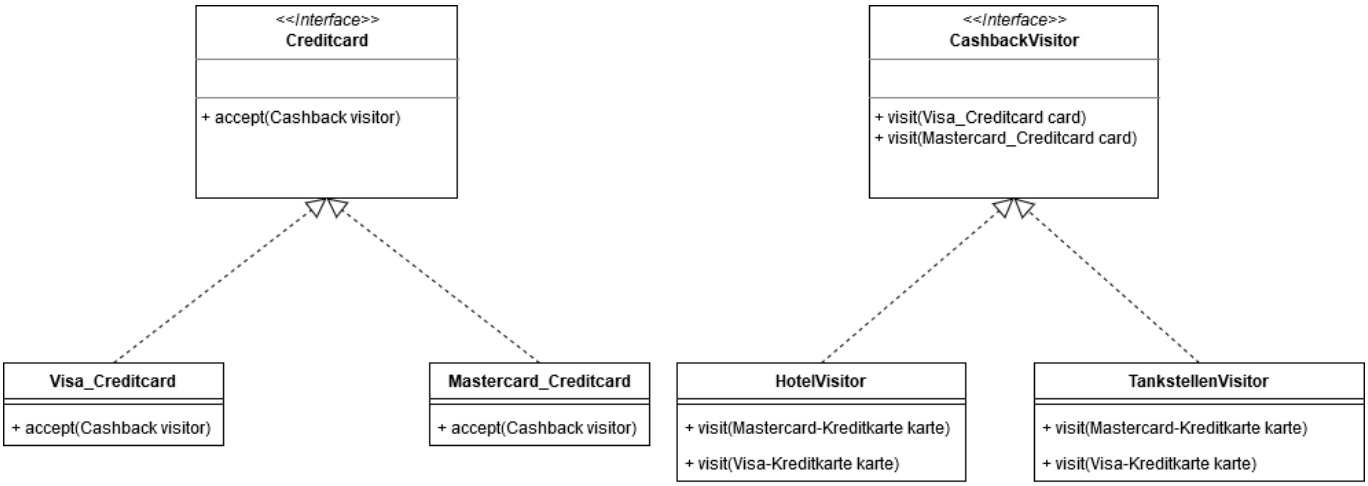

Bei dieser Lösung gibt es ein Kreditkarten Interface, welches Methoden zum Berechnen des Cashbacks vorgibt. Diese Methoden werden direkt von der jeweiligen Kreditkartenklasse implementiert. Das große Problem dieses Ansatzes ist die schlechte Wartbarkeit. In diesem Beispiel haben wir nur 2 Kreditkartentypen. In der Praxis haben Banken jedoch noch Verträge mit weiteren Kreditkartenfirmen, welche wiederum mehrere Kreditkartentypen wie z.B. eine Classic und eine Gold Kreditkarte. Es kann also sehr viele Kreditkartenklassen geben. Ein Problem bekommen wir nun wenn die Bank eine zusätzliche Cashbackaktion startet und wir daher noch eine dritte Methode zur Berechnung des Cashbacks zum Interface Kreditkarte hinzufügen müssen. Denn sobald wir das Interface ändern werden alle Klassen, welche es implementieren unbrauchbar. Damit verstoßen wir auch gegen das Open-Closed-Prinzip.

### Open-Closed-Prinzip

Eine API soll offen für Erweiterungen aber geschlossen gegenüber Änderungen sein.

Wir benötigen also eine Möglichkeit die Funktionalität unserer Kreditkartenklassen zu erweitern ohne Änderungen am Interface(der API) bzw. den Datenklassen vorzunehmen.

## Lösung mit Visitor Pattern




Wenn wir nun das Visitor Pattern anwenden so werden deutliche Änderungen vorgenommen. Das Kreditkarten Interface besitzt keine Methoden mehr um etwas direkt zu berechnen. Stattdessen besitzt es nur noch eine Methode, welche ein Visitorobjekt annimmt. Das Visitorobjekt muss für jede konkrete Datenklasse (in diesem Fall Kreditkarten) eine eigene visit() Methode besitzen. Java unterstützt nur Single Dispatch, das Visitorpattern wird verwendet um Double Dispatch zu simulieren. Daher wenn ein Programm das Cashback für ein Hotel welches mit einer Visa Kreditkarte bezahlt wird berechnen soll so ist daran nicht nur das Visa_Creditcard objekt beteiligt, sondern sowohl Visa_Creditcard als auch HotelVisitor. In der Praxis wird zuerst die accept() Methode der jeweiligen Kreditkarte mit einem passenden Visitor aufgerufen. Diese ruft die visit() Methode des Visitors auf. Da die visit() Methode des Visitor-Interfaces überladen ist wird automatisch die passende visit() Methode für die jeweilige Kreditkarte verwendet. Durch diesen kleinen Umweg bleiben beide beteiligten Objekte nachvollziehbar. Dies ist bei Single Dispatch nicht der Fall.

## Single Dispatch

Basierend auf dem Typ des Objektes welches die Methode enthält wird passende Methode aufgerufen.

## Double Dispatch

Basierend auf dem Typ des Objektes welches die Methode enthält wird passende Methode aufgerufen. Zusätzlich ist macht es jedoch auch noch einen Unterschied, von welchem Objekt die Methode aufgerufen worden ist. Da Java kein Double Dispatch unterstützt wird dieses mithilfe des Visitor Patterns simuliert.

## Wann wird das Visitor Pattern angewannt?

Das Visitor Pattern wird bei Datenklassen verwendet dessen Funktialität leicht erweitert werden soll und wenn eine Simulation des sogenannten Double Dispatch notwendig ist.

## Anwendung in einem Framework

Das bekannte Java-Framework Spring verwendet unter anderem die Klasse org.springframework.beans.factory.config.BeanDefinitionVisitor um die sehr komplexe Konfiguration des Konfigurierens der notwendigen Java Beans elegant zu lösen. Der BeanDefinitionVisitor wird verwendet, um Bean-Metadaten zu analysieren und sie in String oder Object aufzulösen.

## Vollständiges UML

Github