



	\$0	\$100	\$500	
	Bronze	Silver	Gold	CREDIT CARD
Gas	10%	20%	5%	
Food	20%	30%	10%	
Hotel	10%	20%	30%	
Other	10%	5%	20%	

```
class BronzeCreditCard implements CreditCard {
    accept(offerVisitor) {
        offerVisitor.visitBronze(this);
    }
}
```

```
class HotelOfferVisitor implements Visitor {
    visitBronze(bronzeCreditCard) {
        // cashback
    }
    visitSilver(silverCreditCard) {
        // cashback
    }
}
```

```
visitor = new HotelOfferVisitor();
bronzeCC = new BronzeCreditCard();
bronzeCC.accept(visitor); // double dispatch
```

```
class BronzeCard {  
    accept(vistor) {  
        visitor.visitBronze(this);  
    }  
}  
  
class SilverCard {  
    accept(vistor) {  
        visitor.visitSilver(this);  
    }  
}  
  
class HotelVistor {  
    visitSilver(silverCard) {  
        give 5% cashback  
    }  
    visitBronze(bronzeCard) {  
        give 1% cashback  
    }  
}  
  
class GasVistor {  
    visitSilver(silverCard) {  
        give 2% cashback  
    }  
    visitBronze(bronzeCard) {  
        give 0% cashback  
    }  
}
```

The diagram illustrates the Visitor design pattern. It shows four classes: `BronzeCard`, `SilverCard`, `HotelVistor`, and `GasVistor`. `BronzeCard` and `SilverCard` are the elements being visited, while `HotelVistor` and `GasVistor` are the concrete visitors. The `accept` method in `BronzeCard` calls `visitor.visitBronze(this)`, and the `accept` method in `SilverCard` calls `visitor.visitSilver(this)`. The `visitBronze` and `visitSilver` methods in the visitor classes perform the actual visit logic (e.g., giving cashback). Red arrows indicate the flow of control from the element's `accept` method to the corresponding visit method in the visitor. Green arrows indicate the delegation from the element's `accept` method to the visitor's `visit` method.