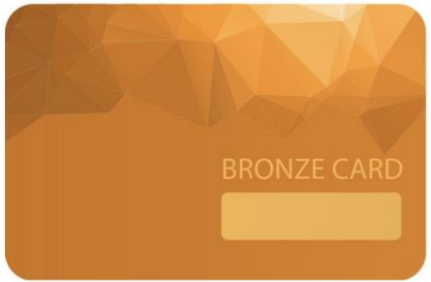
The visitor design pattern

What is it?

A behavioural design pattern that allows the addition of new operations to existing object structures without having to modify the objects themselves.

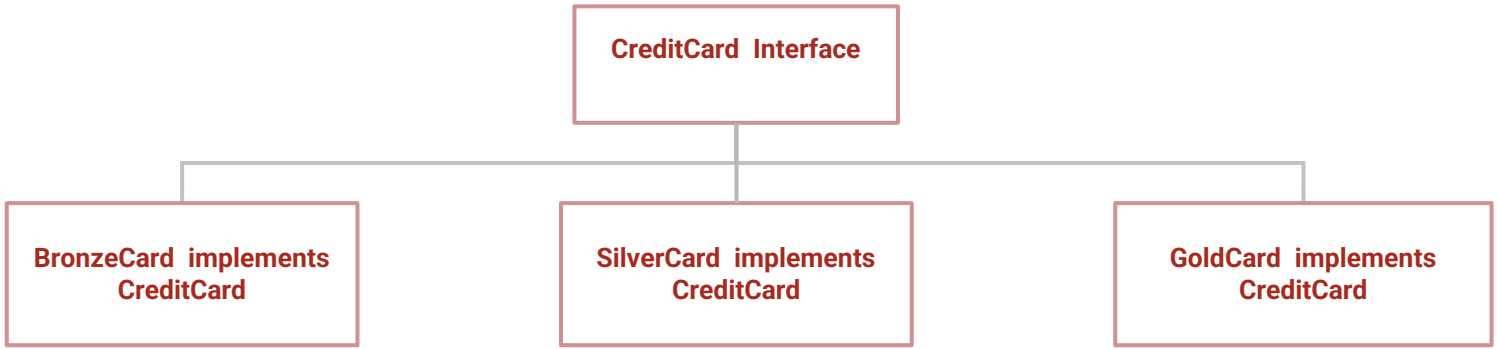
The credit card problem

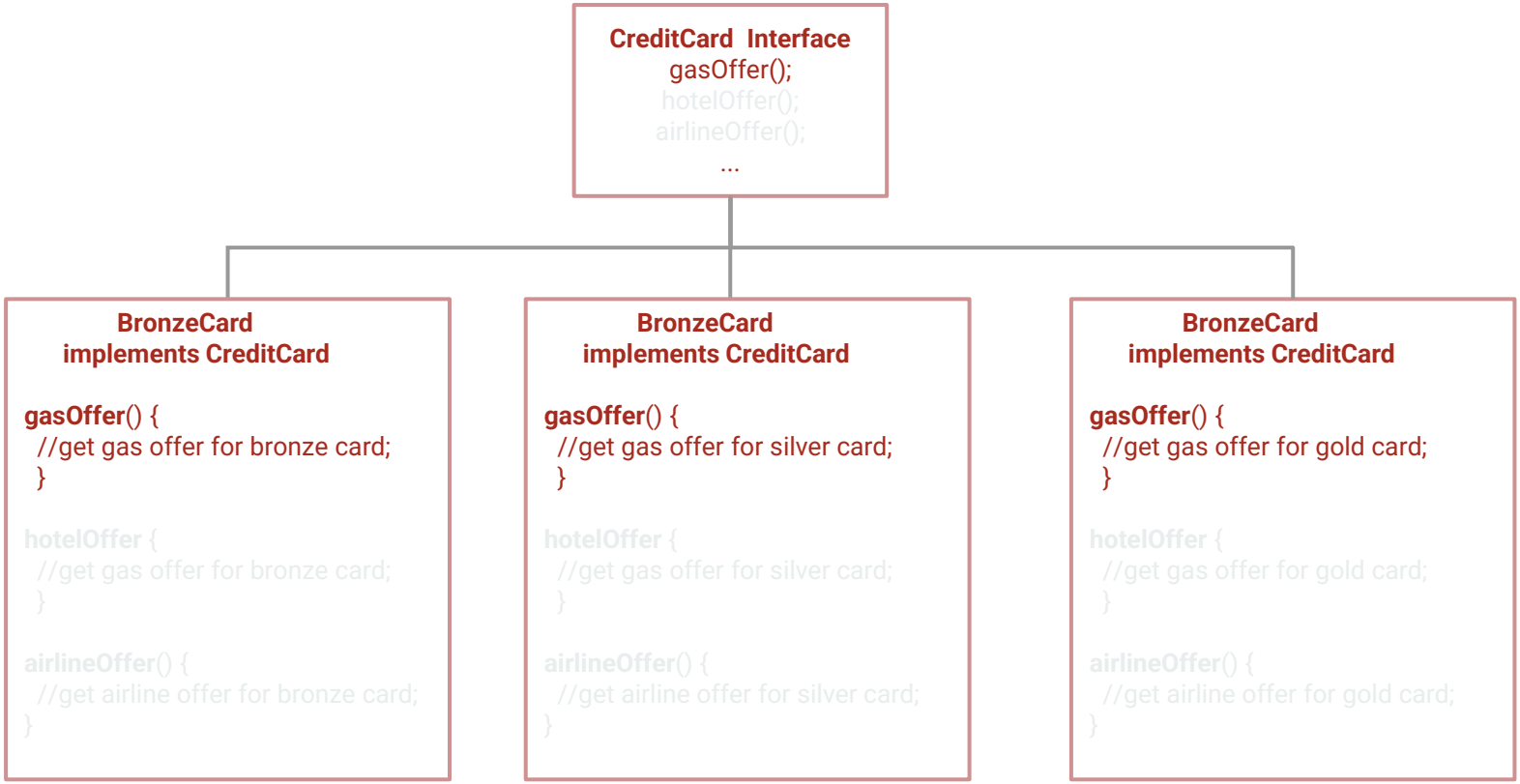
Established bank offers 3 types of credit cards: Bronze, Silver, and Gold.

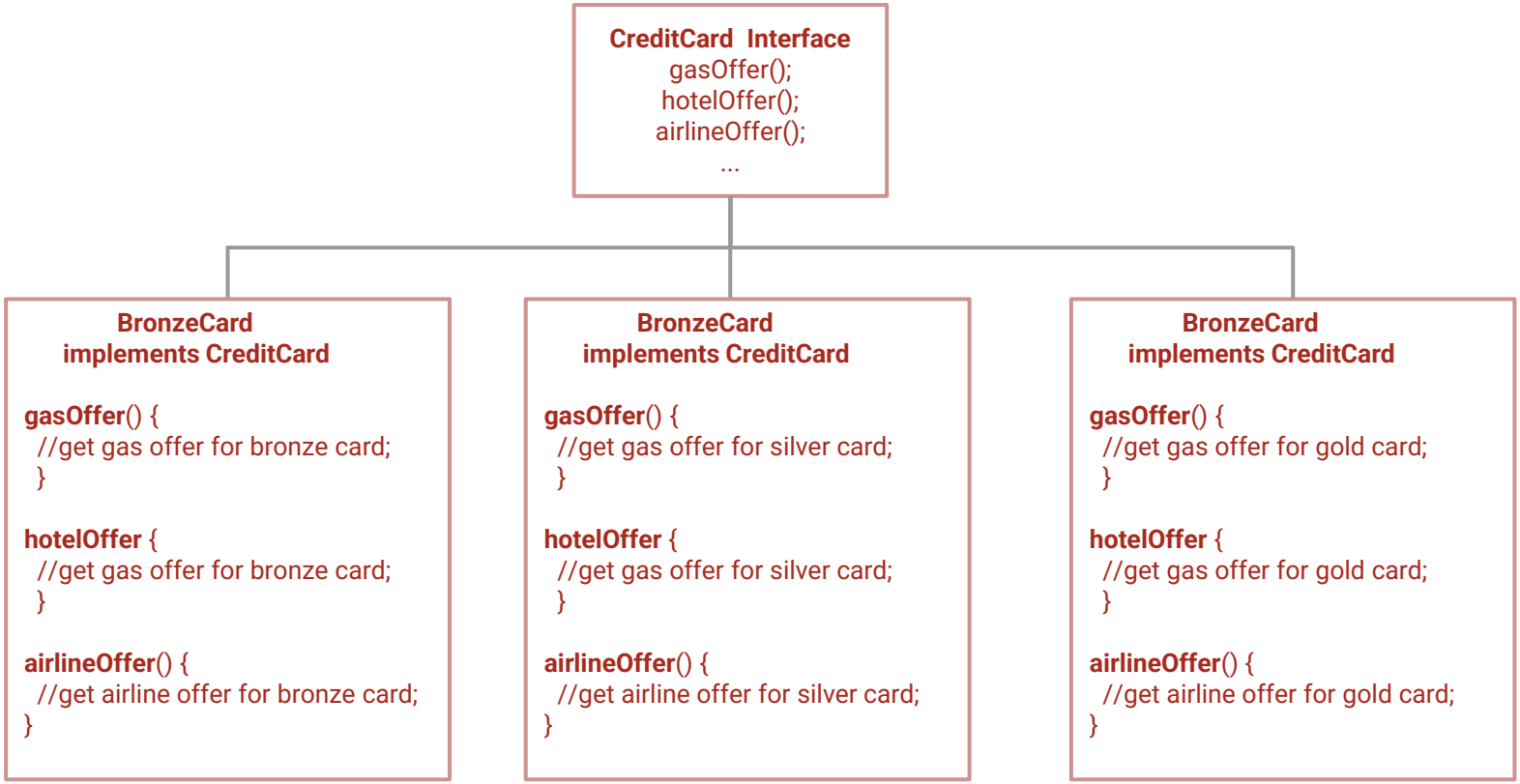


The credit card problem

The bank signs new contracts with 3 companies: a hotel, an airline, and a gas station based on which they will offer cashbacks to customers that use their credit cards.


CreditCard Interface
BronzeCard implements CreditCard
SilverCard implements CreditCard
GoldCard implements CreditCard

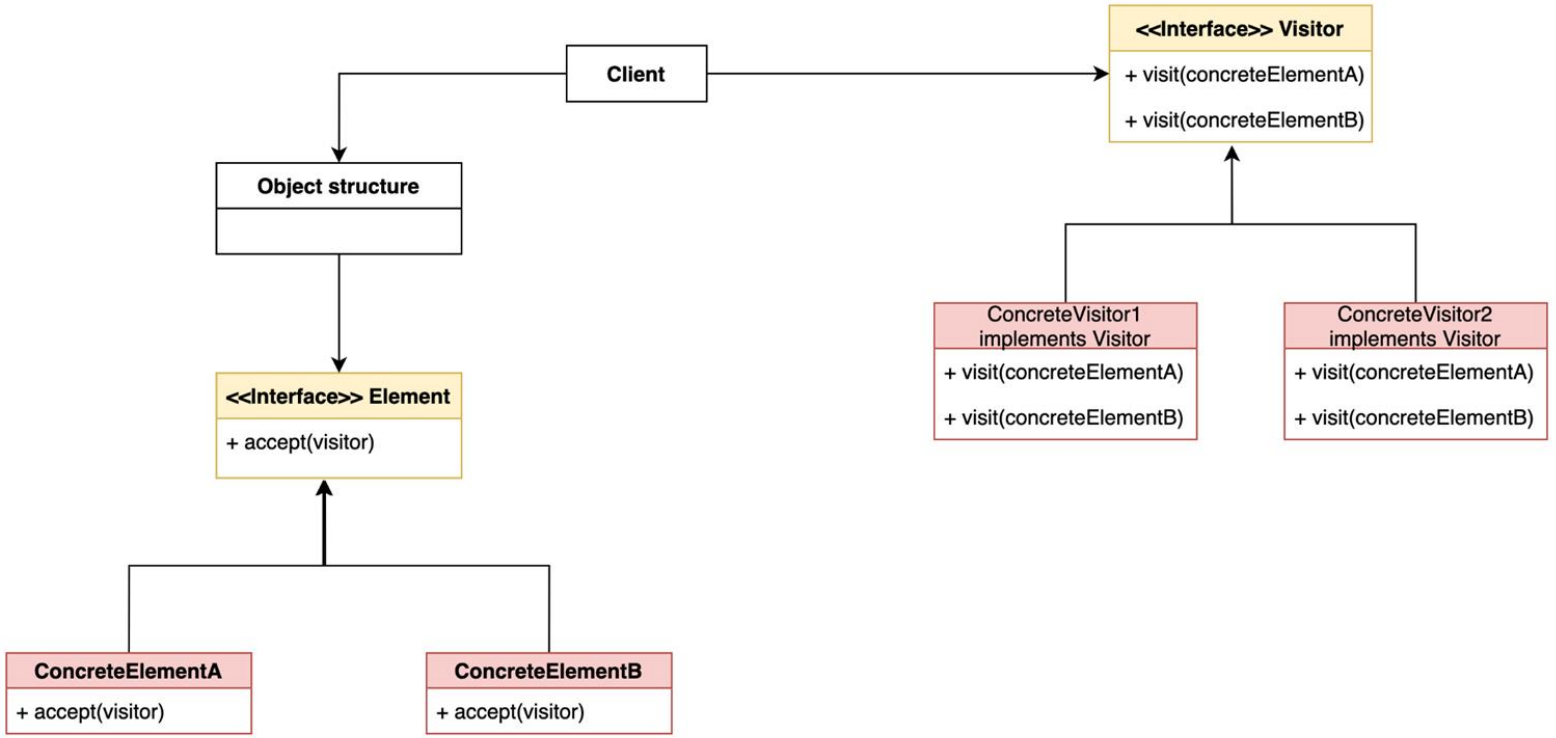

CreditCard Interface
gasOffer();
hotelOffer();
airlineOffer();
...
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for bronze card;
}
hotelOffer {
//get gas offer for bronze card;
}
airlineOffer() {
//get airline offer for bronze card;
}
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for silver card;
}
hotelOffer {
//get gas offer for silver card;
}
airlineOffer() {
//get airline offer for silver card;
}
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for gold card;
}
hotelOffer {
//get gas offer for gold card;
}
airlineOffer() {
//get airline offer for gold card;
}


CreditCard Interface
gasOffer();
hotelOffer();
airlineOffer();
...
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for bronze card;
}
hotelOffer {
//get gas offer for bronze card;
}
airlineOffer() {
//get airline offer for bronze card;
}
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for silver card;
}
hotelOffer {
//get gas offer for silver card;
}
airlineOffer() {
//get airline offer for silver card;
}
BronzeCard
implements CreditCard
gasOffer() {
//get gas offer for gold card;
}
hotelOffer {
//get gas offer for gold card;
}
airlineOffer() {
//get airline offer for gold card;
}

Simple, but ineffective

- We are tightly coupling 2 concepts: the credit cards and external offers.
- We are violating the Open/Closed principle: we should keep our objects open for extensibility, but closed for modification.
- Makes the codebase difficult to maintain.
- Easy to introduce bugs.

Visitor pattern general design



The OfferVisitor interface

```java
package com.company.Offers;

import com.company.CreditCards.BronzeCard;
import com.company.CreditCards.GoldCard;
import com.company.CreditCards.SilverCard;

public interface OfferVisitor {
    void visit(BronzeCard bronzeCard);
    void visit(SilverCard silverCard);
    void visit(GoldCard goldCard);
}
```

The GasOfferVisitor

```java
import com.company.CreditCards.BronzeCard;
import com.company.CreditCards.GoldCard;
import com.company.CreditCards.SilverCard;

public class GasOfferVisitor implements OfferVisitor {
    @Override
    public void visit(BronzeCard bronzeCard) {
        System.out.println("Computing the gas offer for the Bronze Card");
    }

    @Override
    public void visit(SilverCard silverCard) {
        System.out.println("Computing the gas offer for the Silver Card");
    }

    @Override
    public void visit(GoldCard goldCard) {
        System.out.println("Computing the gas offer for the Gold Card");
    }
}
```

The HotelOfferVisitor

```java
import com.company.CreditCards.BronzeCard;
import com.company.CreditCards.GoldCard;
import com.company.CreditCards.SilverCard;

public class HotelOfferVisitor implements OfferVisitor{
    @Override
    public void visit(BronzeCard bronzeCard) {
        System.out.println("Computing the hotel offer for the Bronze Card");
    }

    @Override
    public void visit(SilverCard silverCard) {
        System.out.println("Computing the hotel offer for the Silver Card");
    }

    @Override
    public void visit(GoldCard goldCard) {
        System.out.println("Computing the hotel offer for the Gold Card");
    }
}
```

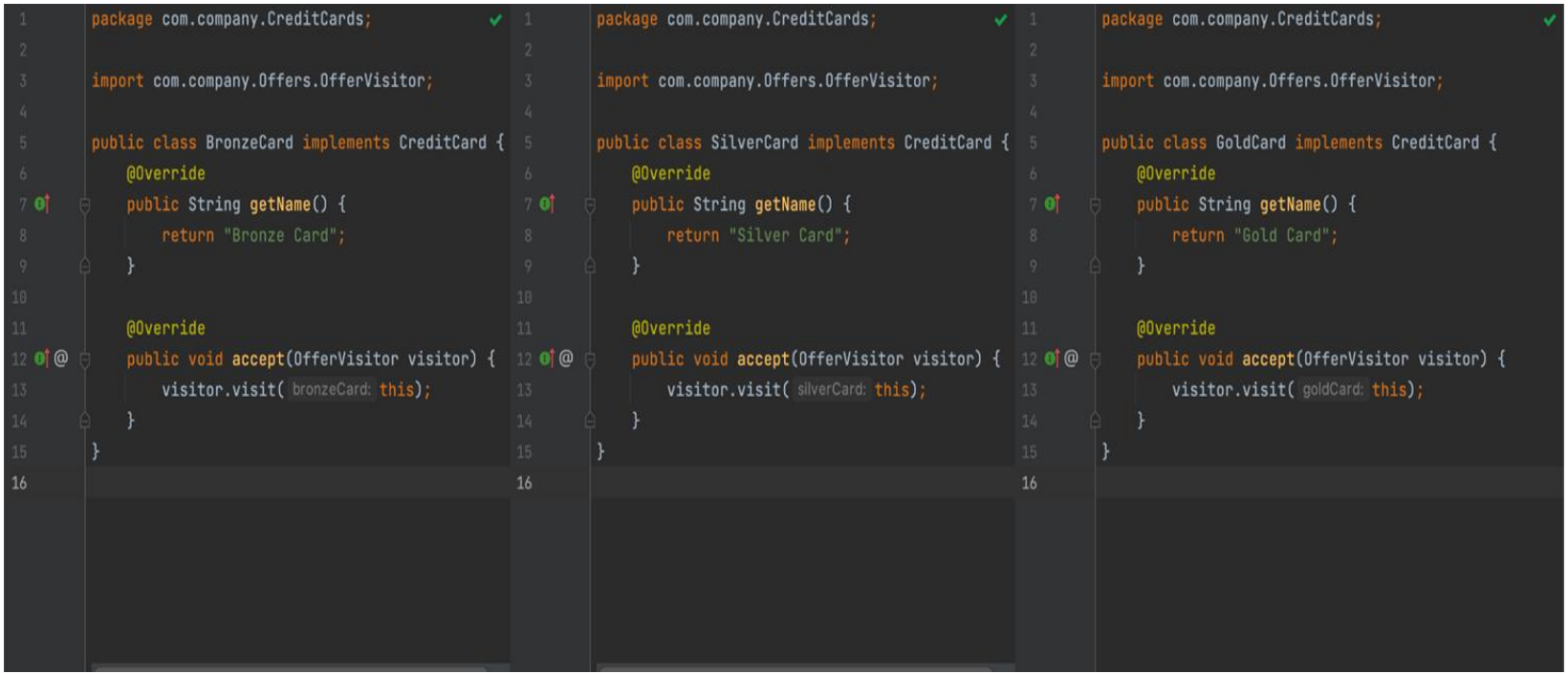
The CreditCard Interface

```java
package com.company.CreditCards;

import com.company.Offers.OfferVisitor;

public interface CreditCard {
    String getName();
    void accept(OfferVisitor visitor);
}
```

The BronzeCard, SilverCard, and GoldCard classes



```java
package com.company.CreditCards;

import com.company.Offers.OfferVisitor;

public class BronzeCard implements CreditCard {
    @Override
    public String getName() {
        return "Bronze Card";
    }

    @Override
    public void accept(OfferVisitor visitor) {
        visitor.visit( bronzeCard: this);
    }
}
```

```java
package com.company.CreditCards;

import com.company.Offers.OfferVisitor;

public class SilverCard implements CreditCard {
    @Override
    public String getName() {
        return "Silver Card";
    }

    @Override
    public void accept(OfferVisitor visitor) {
        visitor.visit( silverCard: this);
    }
}
```

```java
package com.company.CreditCards;

import com.company.Offers.OfferVisitor;

public class GoldCard implements CreditCard {
    @Override
    public String getName() {
        return "Gold Card";
    }

    @Override
    public void accept(OfferVisitor visitor) {
        visitor.visit( goldCard: this);
    }
}
```

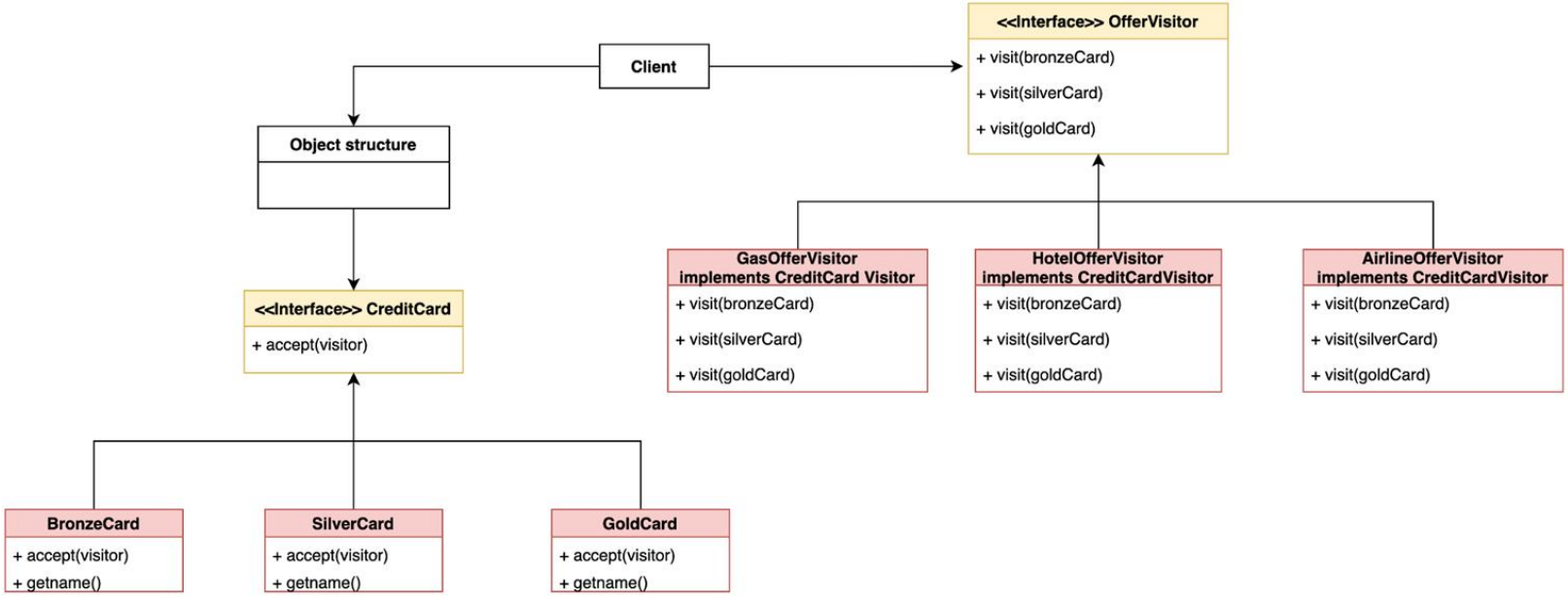
Running the example

```
11
12  public class Main {
13
14      public static void main(String[] args) {
15          CreditCard bronzeCard = new BronzeCard();
16          CreditCard silverCard = new SilverCard();
17          CreditCard goldCard = new GoldCard();
18
19          OfferVisitor gasOfferVisitor = new GasOfferVisitor();
20          bronzeCard.accept(gasOfferVisitor);
21          silverCard.accept(gasOfferVisitor);
22          goldCard.accept(gasOfferVisitor);
23
24          OfferVisitor airlineOfferVisitor = new AirlineOfferVisitor();
25          bronzeCard.accept(airlineOfferVisitor);
26
27          OfferVisitor hotelOfferVisitor = new HotelOfferVisitor();
28          goldCard.accept(hotelOfferVisitor);
29      }
30  }
31
```

```
Run:  Main
/Library/Java/JavaVirtualMachines/jdk-16.0.1.jdk/Contents/Home/bin/java -javaagent:/Applications/IntelliJ IDEA CE.app/Contents/lib/idea_rt.jar
Computing the gas offer for the Bronze Card
Computing the gas offer for the Silver Card
Computing the gas offer for the Gold Card
Computing the airline offer for the Bronze Card
Computing the hotel offer for the Gold Card

Process finished with exit code 0
```

Credit card solution diagram



PROS

- Easy to add new operations to objects without modifying them.
- Related behaviour is focused in a single concrete visitor.
- It implements double dispatching in languages that don’t support it natively.

CONS

- Your Visitor can modify your Elements since an instance of the Element is sent to the Visitor.
- Code can become less readable.
- It requires a new Visitor class for every action.

Questions?

Thank you!