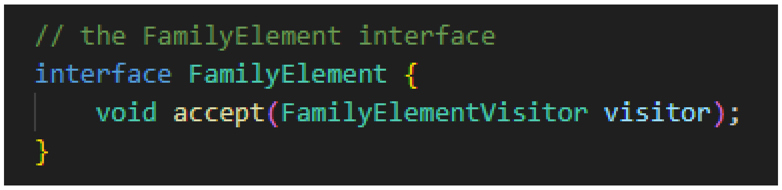
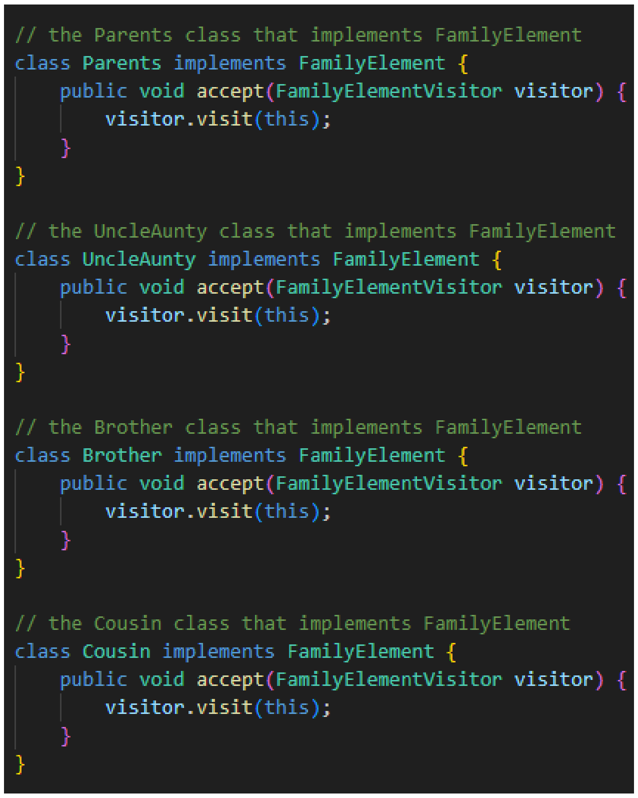
The Visitor Pattern is like a car wash for cars. When you take your car to a car wash, the car wash can wash, wax, or vacuum your car without changing the car itself. The Visitor Pattern is like that, but for computer programs. It lets you add new things to a program without changing the program itself. This makes it easier to change and update the program over time. If I would not use the visitor pattern then I would face a lot of issues, like changing the program all the time when I want to add something new, which means that the code is separated. When you would decide to change something then you would just need to change the classes and not the client code.



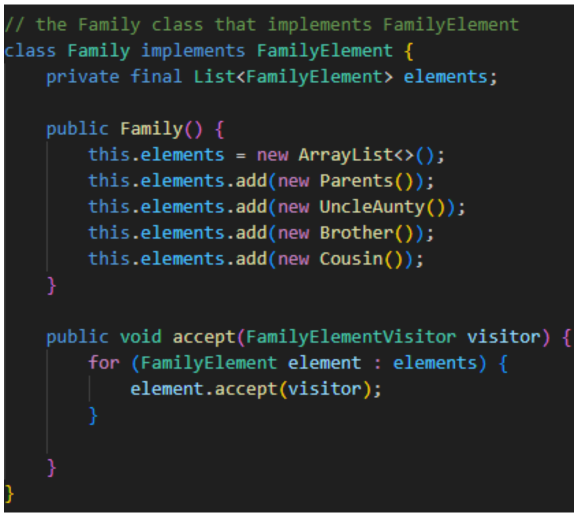
```
// the FamilyElement interface
interface FamilyElement {
    void accept(FamilyElementVisitor visitor);
}
```

This is the FamilyElement interface which is created to represent the elements in the family hierarchy.
The accept method is declared and will accept FamilyElementVistor.
All the elements in the family hierarchy are responsible for accepting a visitor. The visitor is responsible to do operations on these elements.
This interface is designed to establish a common ground for all family elements.



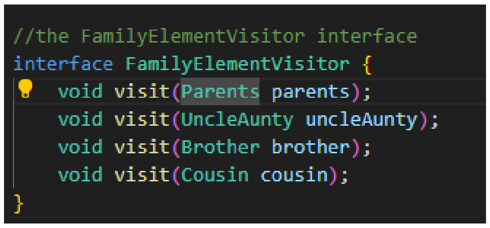
```
// the Parents class that implements FamilyElement
class Parents implements FamilyElement {
    public void accept(FamilyElementVisitor visitor) {
        visitor.visit(this);
    }
}

// the UncleAunty class that implements FamilyElement
class UncleAunty implements FamilyElement {
    public void accept(FamilyElementVisitor visitor) {
        visitor.visit(this);
    }
}

// the Brother class that implements FamilyElement
class Brother implements FamilyElement {
    public void accept(FamilyElementVisitor visitor) {
        visitor.visit(this);
    }
}

// the Cousin class that implements FamilyElement
class Cousin implements FamilyElement {
    public void accept(FamilyElementVisitor visitor) {
        visitor.visit(this);
    }
}
```

These are the Concrete Element CLasses.(Parents, UncleAunty, Brother, Cousin)
They all implement the FamilyElement Interface. They all have the accept method which will then call the visitor method of the FamilyElementVisitor.



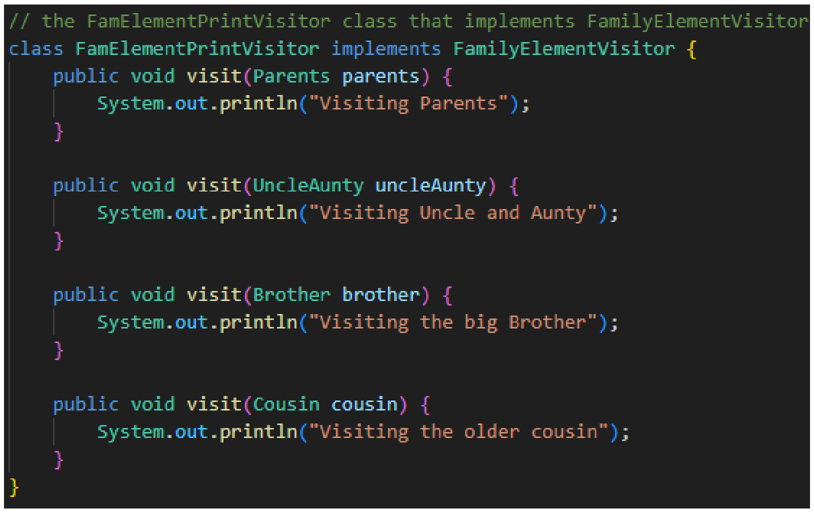
```java
// the Family class that implements FamilyElement
class Family implements FamilyElement {
    private final List<FamilyElement> elements;

    public Family() {
        this.elements = new ArrayList<>();
        this.elements.add(new Parents());
        this.elements.add(new UncleAunty());
        this.elements.add(new Brother());
        this.elements.add(new Cousin());
    }

    public void accept(FamilyElementVisitor visitor) {
        for (FamilyElement element : elements) {
            element.accept(visitor);
        }

    }
}
```

This is the family class which implements the interface FamilyElement.
This class has the collection of all the family elements.
In the constructor it creates instances of different family elements and adds them to the elements list.
There is also a for-each loop which is responsible to simplify the iteration over the elements in a collection in this case I have a list.
This represents each element in the list during the iteration.
When the loop iterates over each element in the elements list then the accept method is also called.



```java
//the FamilyElementVisitor interface
interface FamilyElementVisitor {
    void visit(Parents parents);
    void visit(UncleAunty uncleAunty);
    void visit(Brother brother);
    void visit(Cousin cousin);
}
```

The interface FamilyElementVisitor declares the visit methods for each family element.
Each visit method in this interface represents an operation that the visitor can perform on a specific type of the family element.



```java
// the FamElementPrintVisitor class that implements FamilyElementVisitor
class FamElementPrintVisitor implements FamilyElementVisitor {
    public void visit(Parents parents) {
        System.out.println("Visiting Parents");
    }

    public void visit(UncleAunty uncleAunty) {
        System.out.println("Visiting Uncle and Aunty");
    }

    public void visit(Brother brother) {
        System.out.println("Visiting the big Brother");
    }

    public void visit(Cousin cousin) {
        System.out.println("Visiting the older cousin");
    }
}
```

This class name is FamElementPrintVisitor implements the FamilyElementVisitor interface. This class knows exactly which family member I want to visit and what to say when we visit them.

There are many benefits of using the Visitor pattern like Separations of Concerns and extensibility.
With Separation of Concerns I mean that, it separates the operations in this case visitors from the element being visited. Each visitor is responsible for a specific operation.
If I did not use this pattern then I would have a tightly coupled code, where I would put all the operations related to different elements directly into those elements classes.
Extensibility is also a big factor of this pattern. I can easily add new operations without modifying the existing elements. New elements can be added without changing the structure of the visited elements. WIthout this pattern then I would need to violate the Open/Close principle because adding new operations would require me to modify the existing elements classes.