Visitor pattern

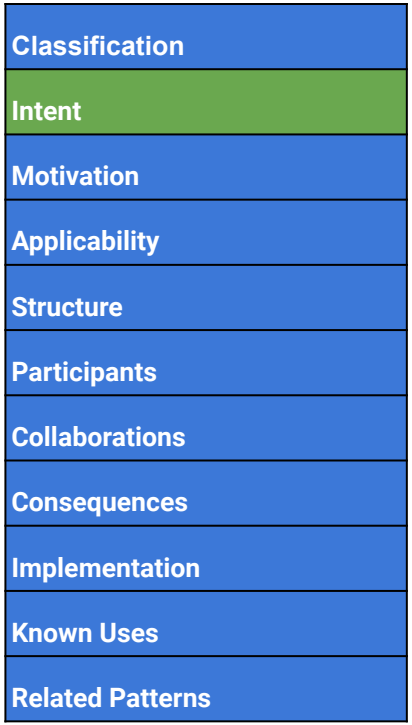
Terēza Lakstīgala
Krišjānis Bergmanis

Classification

Classification
Intent
Motivation
Applicability
Structure
Participants
Collaborations
Consequences
Implementation
Known Uses
Related Patterns

Pattern Name	Visitor
Classification	Object, Behavioral
Also known as	-

Intent



Ļauj pievienot objektam funkcijas, neizmainot šo objektu

Motivation

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

Problem

Kad objekts sastāv no daudzām nesaistītām klasēm un ir nepieciešams lietot jaunu funkcionalitāti regulāri, tad nav izdevīgi visu laiku pievoniet jaunas apakšklases, jo izmētājot funkcionalitāti pa visām klasēm padara kodu nepārlasāmu un grūti uzturamu, un maināmu

Motivation

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

Solution

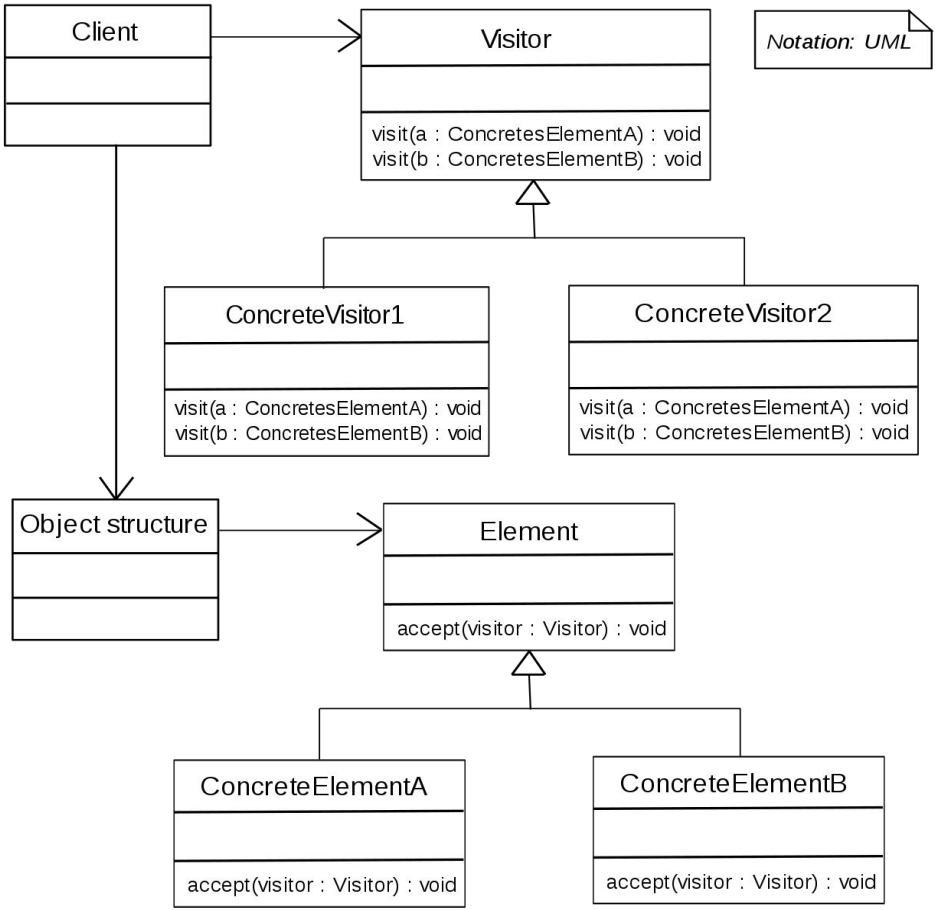
- Definēt (visitor) objektu kas implementē funkciju kuru vēlāk izmantot citā objektā (Elementā)
- Vēlāk lietotājs var izsaukt visit funkciju kas ‘nosūta’ pieprasījumu, kurš tiek pieņemts Elementā. Pēc tam visitor objekts ļauj izpildīt padotās funkcijas šim Elementam

Applicability

- Objekta struktūrā ir nepieciešams lietot daudzas nesaustītas funkcijas
- Objekts sastāv no daudzām klasēm, kuras nav paredzēts mainīt
- Bieži jāpievieno jaunas funkcijas
- Algoritmi, kas iekļauj daudzas klases, kuras jpārvalda vienuviet
- Algoritmi kur, vajag sadarbību starp vairākām nesaistītām klasēm

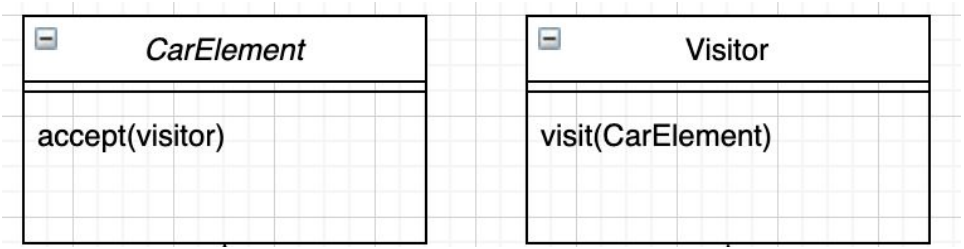
Structure

- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns



Structure

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

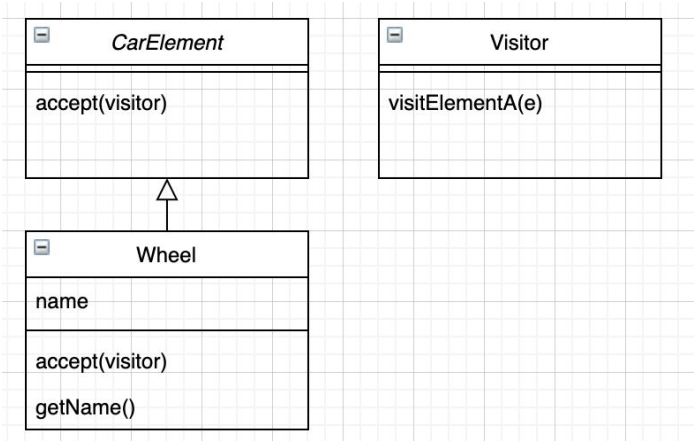



```
interface CarElement {
    void accept(CarElementVisitor visitor);
}

interface CarElementVisitor {
    void visit(Body body);
    void visit(Car car);
    void visit(Engine engine);
    void visit(Wheel wheel);
}
```

Structure

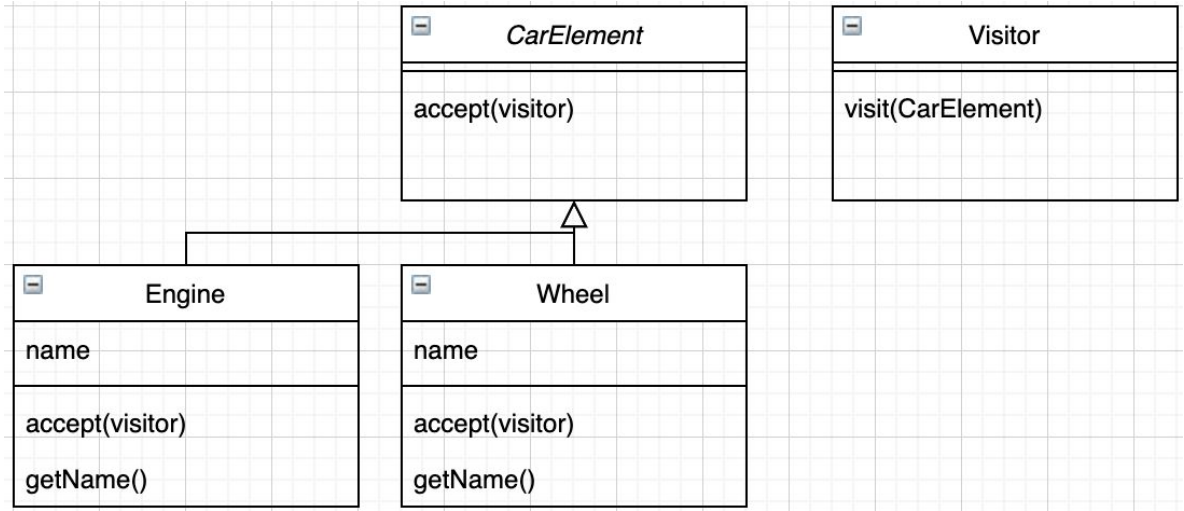
- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

```
class Wheel implements CarElement
  private final String name;

  public Wheel(final String name)
      this.name = name;
  }

  public String getName() {
      return name;
  }

  @Override
  public void accept(CarElementVisitor visitor) {
      visitor.visit(this);
  }
}
```



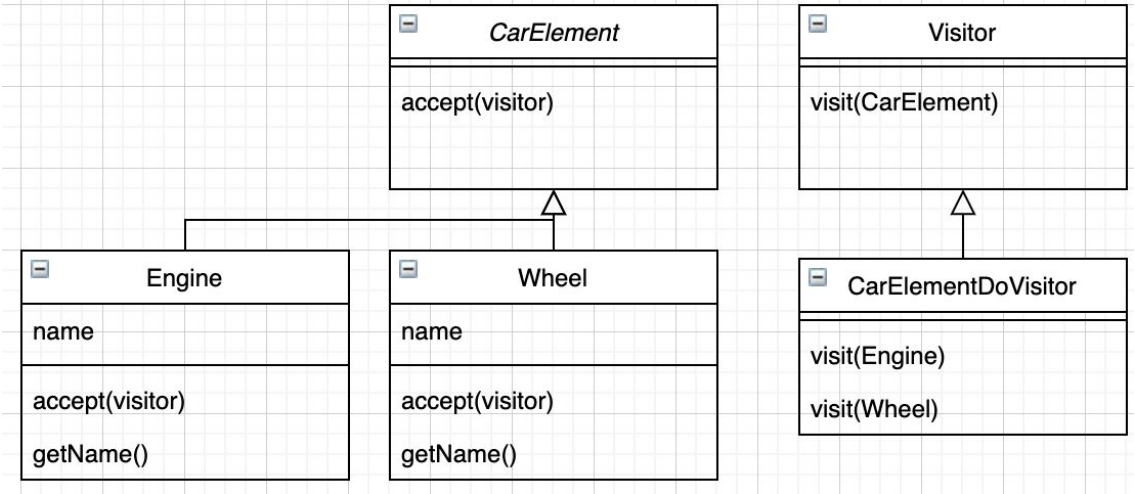
Structure

- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns



Structure

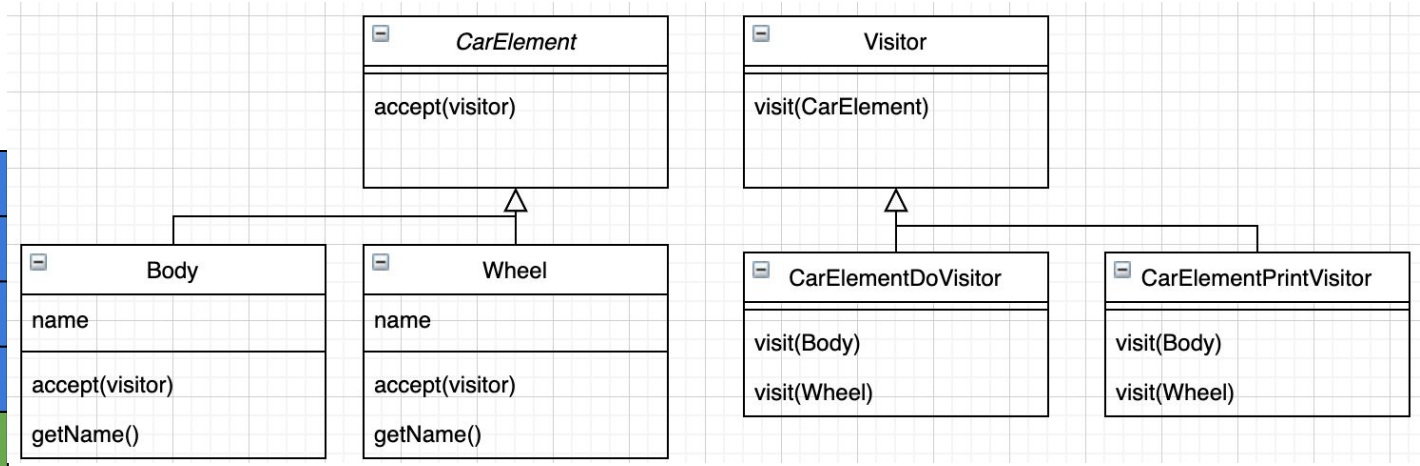
- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns




```
class CarElementDoVisitor implements CarElementVisitor {
    @Override
    public void visit(Body body) {
        System.out.println("Moving my body");
    }

    @Override
    public void visit(Car car) {
        System.out.println("Starting my car");
    }

    @Override
    public void visit(Wheel wheel) {
        System.out.println("Kicking my " + wheel.getName() + " wheel");
    }
```

Structure

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

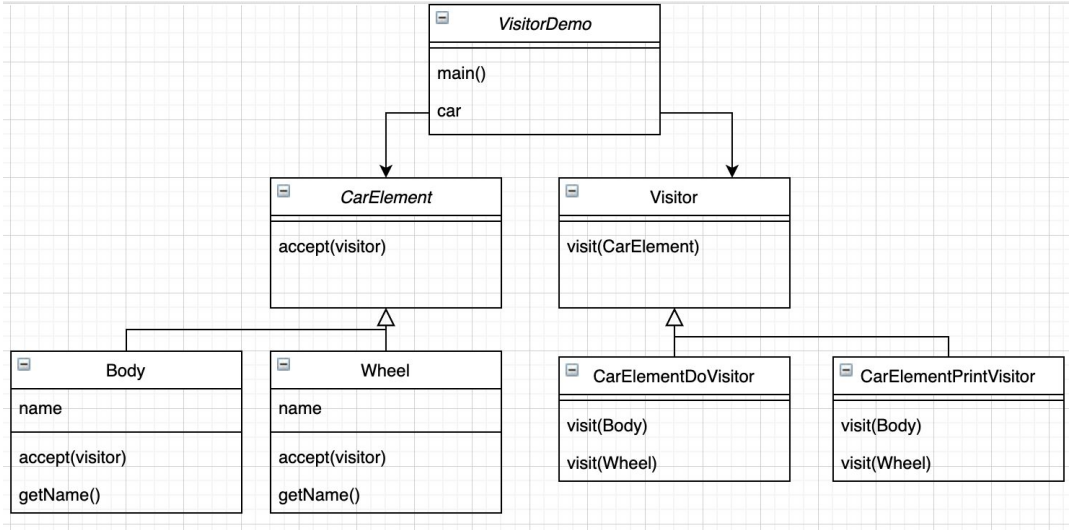



```
class CarElementPrintVisitor implements CarElementVisitor {
    @Override
    public void visit(Body body) {
        System.out.println("Visiting body");
    }

    @Override
    public void visit(Car car) {
        System.out.println("Visiting car");
    }

    @Override
    public void visit(Engine engine) {
        System.out.println("Visiting engine");
    }
```

Structure

- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

```
public class VisitorDemo {
    public static void main(final String[] args) {
        Car car = new Car();

        car.accept(new CarElementPrintVisitor());
        car.accept(new CarElementDoVisitor());
    }
}
```



Structure

- Classification
- Intent
- Motivation
- Applicability
- **Structure**
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

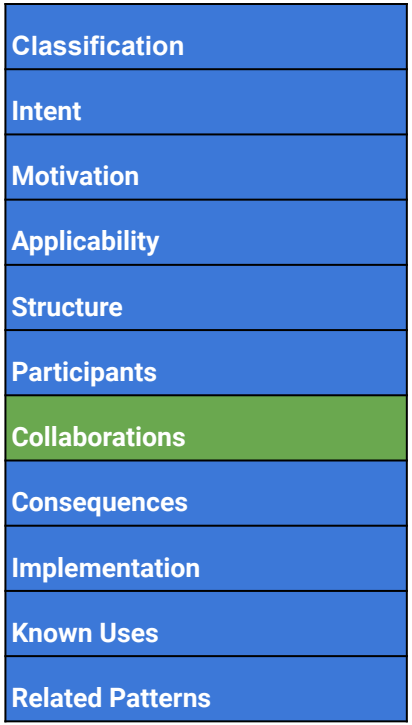
```
Visiting front right wheel
Visiting back left wheel
Visiting Engine
Visiting Body
Kicking front right wheel
Kicking back left wheel
Starting Engine
Moving Body
```

Participants

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

Participant	Role
Visitor base	Declares a set of visiting methods that can take concrete elements of an object structure as arguments
Element base	Declares a method for “accepting” the visitors
Concrete visitors	Implements several versions of the same behaviours tailored for different concrete element classes
Concrete elements	Must implement the acceptance method, purpose of which is to redirect the call to proper visitors method corresponding to the current element class
Client	Usually represents a collection or some other complex object, for example, Composite tree

Collaborations



Client klases objekti nav tieši informēti par konkrētiem elementiem, jo tie sadarbojas caur Interfeisiem

Consequences

- Objektiem var netieši piešķirt jaunu funkcionalitāti
- Vienai un tai pašai klasei var piešķirt dažādas versijas konkrētai funkcijai
- Visitor objekts var apkopot daudz informāciju strādājot ar dažādiem objektiem, kas ļauj apstrādāt sarežģītākas datu struktūras, kā kokus

Implementation

- Deklarēt visitor interfeisu ar visit metodēm
- Deklarēt elemeta interfeisu un implementēt konkrētas klases ar metodēm (ieskaitot accept())
- Elementa klases sastrādājas tikai caur visitor interfeisu
- Katru jaunu funkcionalitāti ko nevar iekļaut Element klasē var izveidot jaunu visitor klasi un tajā implementēt nepieciešamo funkciju
- Jāizveido visitor objekti un tie jāpadod Elementā caur accept() funkcijām

Related Patterns

- Classification
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Known Uses
- Related Patterns

Visitor var tikt izmantots kopā ar Iterator lai iterētu caur datu struktūrām un izpildīt funkcijas ar to Elementiem pat ja šie ir dažādu klašu elementi