The Visitor Pattern

Natalia Greyz

Intent

- Represent an operation to be performed on the elements of an object structure.
- Define a new operation without changing the classes of the elements on which it operates
- Let’s consider an example

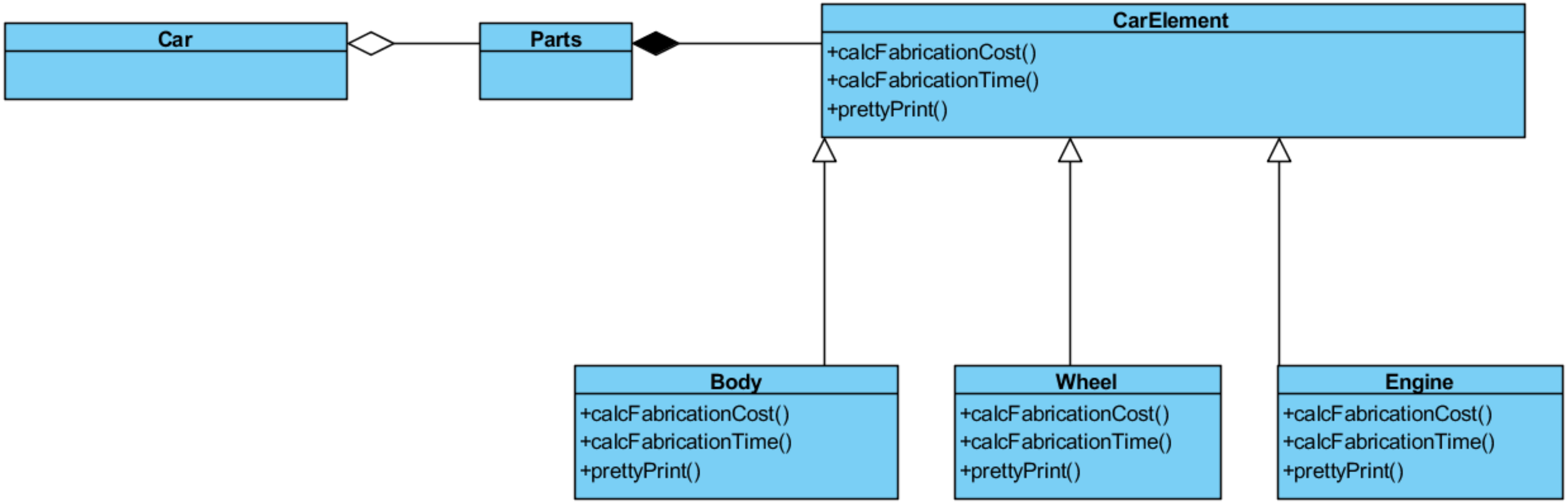
A word from the censor board

- The following presentation has been approved for all audiences
- It does not contain :
  - Terror attacks
  - Crashing planes
  - Fight scenes
  - Heavy weaponry
- But don’t worry, we are going to talk about things that have engines and wheels!

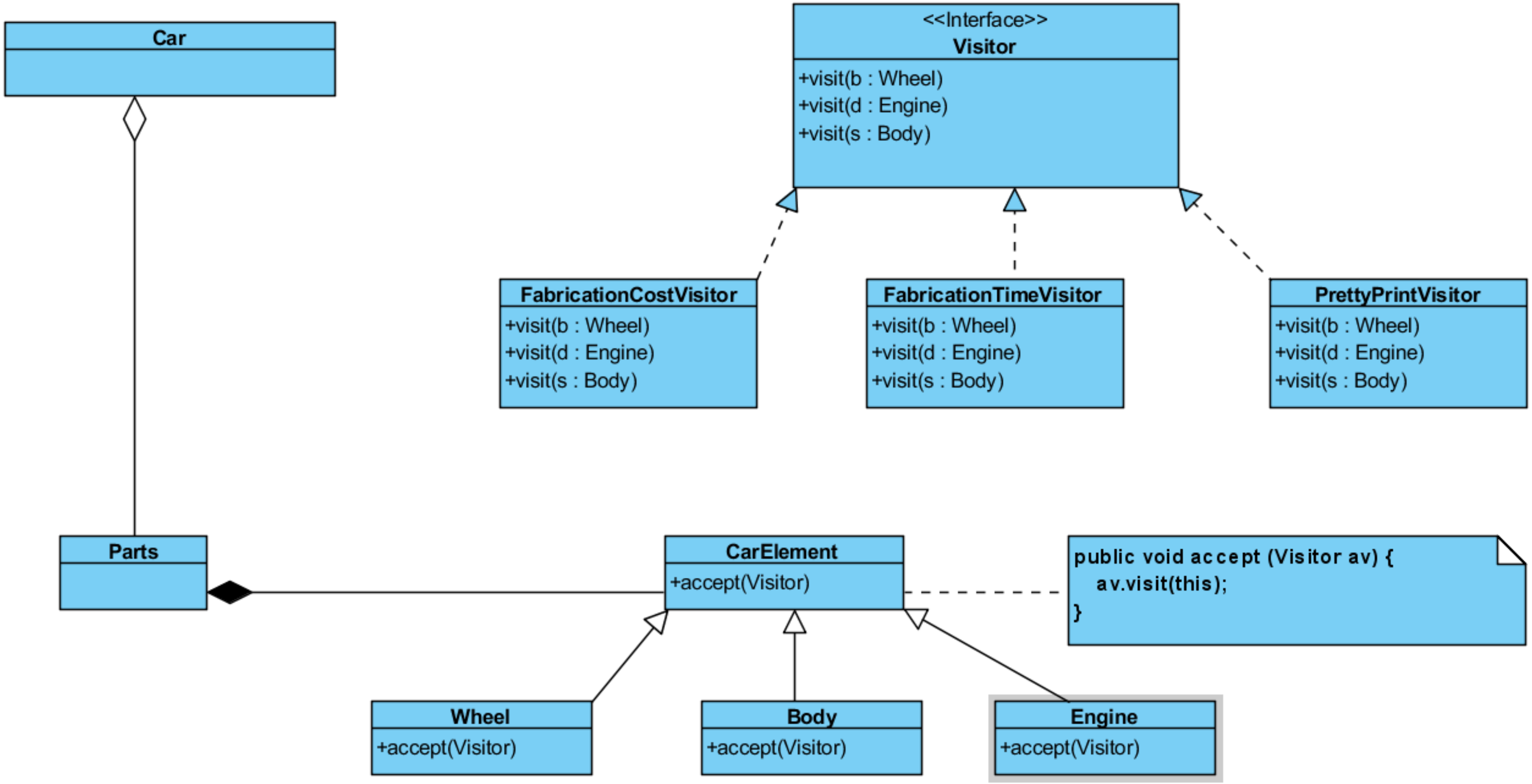
We are going to build a Car!

- The car is a class that comprises car elements of 3 different types :
  - Engine
  - Wheel
  - Body
- Tasks: calculate car fabrication cost, time and print the list of required parts
- In order to perform those tasks, we need to implement the following operations for each one of the parts:
  1. Fabrication cost calculation
  2. Fabrication time calculation
  3. Print information about the part

Design without the Visitor



Let’s apply Visitor pattern



Car

<<Interface>>
Visitor
+visit(b : Wheel)
+visit(d : Engine)
+visit(s : Body)

FabricationCostVisitor
+visit(b : Wheel)
+visit(d : Engine)
+visit(s : Body)

FabricationTimeVisitor
+visit(b : Wheel)
+visit(d : Engine)
+visit(s : Body)

PrettyPrintVisitor
+visit(b : Wheel)
+visit(d : Engine)
+visit(s : Body)

Parts

CarElement
+accept(Visitor)

```
public void accept (Visitor av) {
    av.visit(this);
}
```

Wheel
+accept(Visitor)

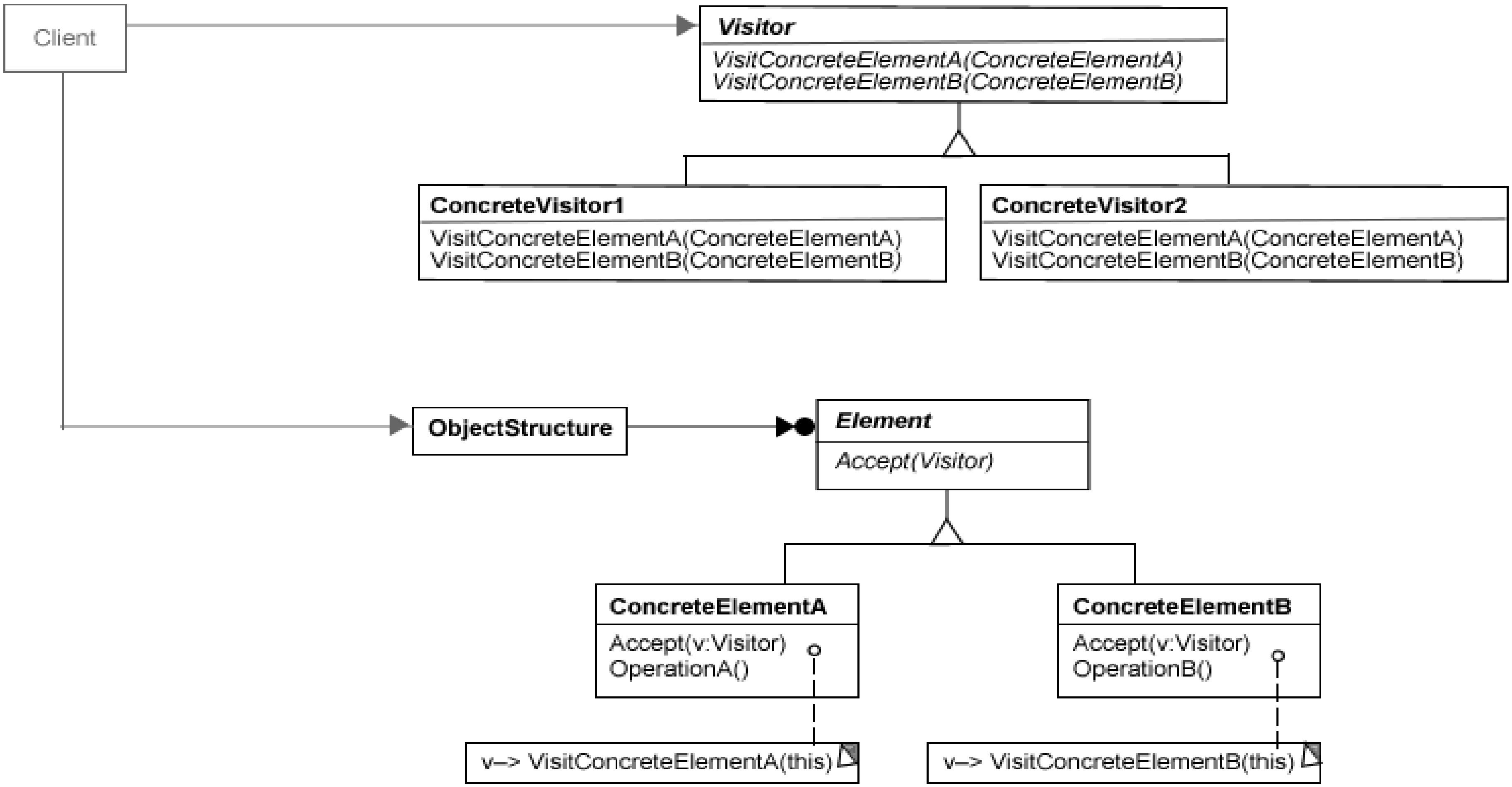
Body
+accept(Visitor)

Engine
+accept(Visitor)

Participants

- Visitor
- Concrete Visitor
- Element
- Concrete Element
- Object Structure

Pattern diagram



Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor)
ConcreteElementA
Accept(v:Visitor)
OperationA()
ConcreteElementB
Accept(v:Visitor)
OperationB()
v–> VisitConcreteElementA(this)
v–> VisitConcreteElementB(this)

Benefits

- The Visitor pattern is a way to separate an algorithm from an object structure it operates on
  - A practical result of this separation is the ability to add new operations to existing object structures without modifying those structures
  - If we want to add a new operation “calcWeight” , we just need to create a new Concrete Visitor “WeightVisitor”
- Gathers related operations and separates unrelated operations

Benefits – cont.

- Visitors can visit objects that don’t have a common parent class
- Visitors can accumulate state as they visit each element
  - This is extremely useful in many cases where the action performed on the object depends on previous such actions.

Issues

- Adding new Concrete Element classes is complicated
  - If we want to add class “SoundSystem”, we need to add operation visit (SoundSystem) to all the Visitors
- Allowing a Visitor to access the internal state of a Concrete Element breaks encapsulation

When to use

- Apply this pattern if you have:

1. An object structure that contains many different/unrelated classes of objects
2. Many distinct and unrelated operations on these objects
    1. Want to avoid “polluting” their classes
3. Classes defining the object structure rarely change, but operations change frequently
    1. You don’t want to modify object every time when a new operation needs to be added

Known applications

- Smalltalk-80 compiler
  - Visitors are used for algorithms that analyze code
- Inventor – a toolkit for developing 3D applications.
  - Represents 3D scene as hierarchy of nodes, each representing a geometric object
  - Visitors are used for operations like rendering or mapping events that require traversing the hierarchy in different ways