Behavioural Design Patterns

Part 5

Behavioural Design Patterns

- State
- Strategy
- Observer
- Command
- Visitor

Behavioural Patterns: Visitor

- Suppose we have a hierarchy of employee classes:
  - `HourlyEmployee`, `SalariedEmployee`, etc.

- We need to be able to run reports on these employees. We may want:
  - A report of the earnings of all hourly employees
  - A report of the earnings of all employees
  - …

Behavioural Patterns: Visitor

- We don’t want to violate the Single Responsibility Principle by mixing reporting code into the employee classes

- We need to be able to add new reports at any given time
    - We don’t want to violate the Open-Closed Principle by having to modify the employee classes later

Behavioural Patterns: Visitor

Design Pattern:

Visitor

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

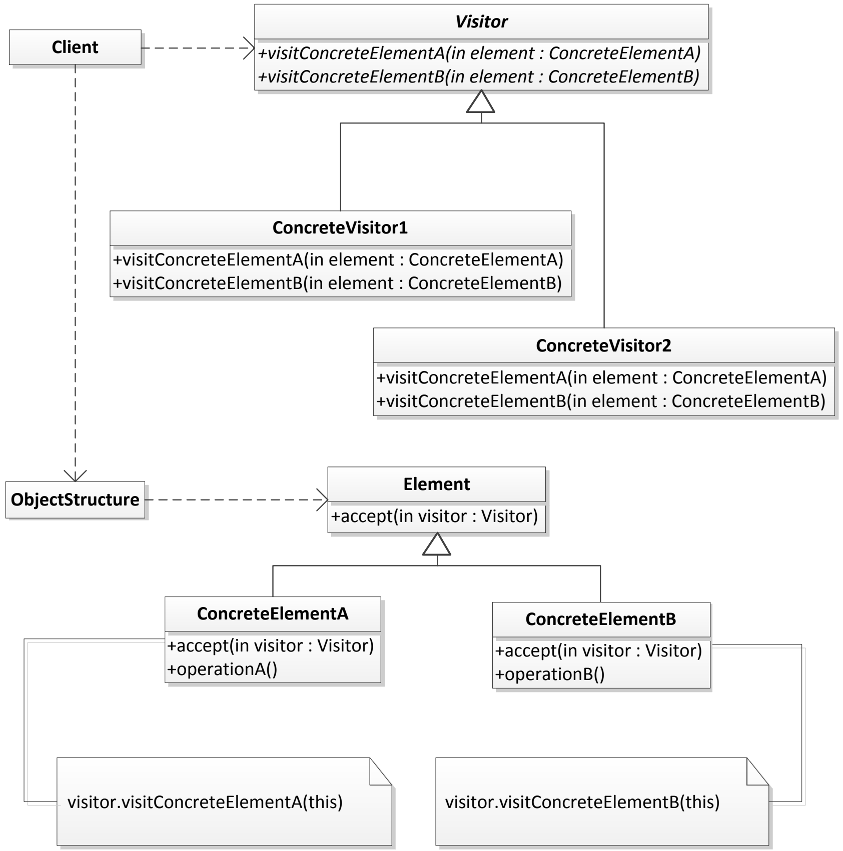
Behavioural Patterns: Visitor

- Applicability:
  - An object structure contains many classes of objects with differing interfaces, and you want to perform operations on these objects that depend on their concrete classes
  - Many distinct and unrelated operations need to be performed on objects in an object structure, and you want to avoid polluting their classes with these operations

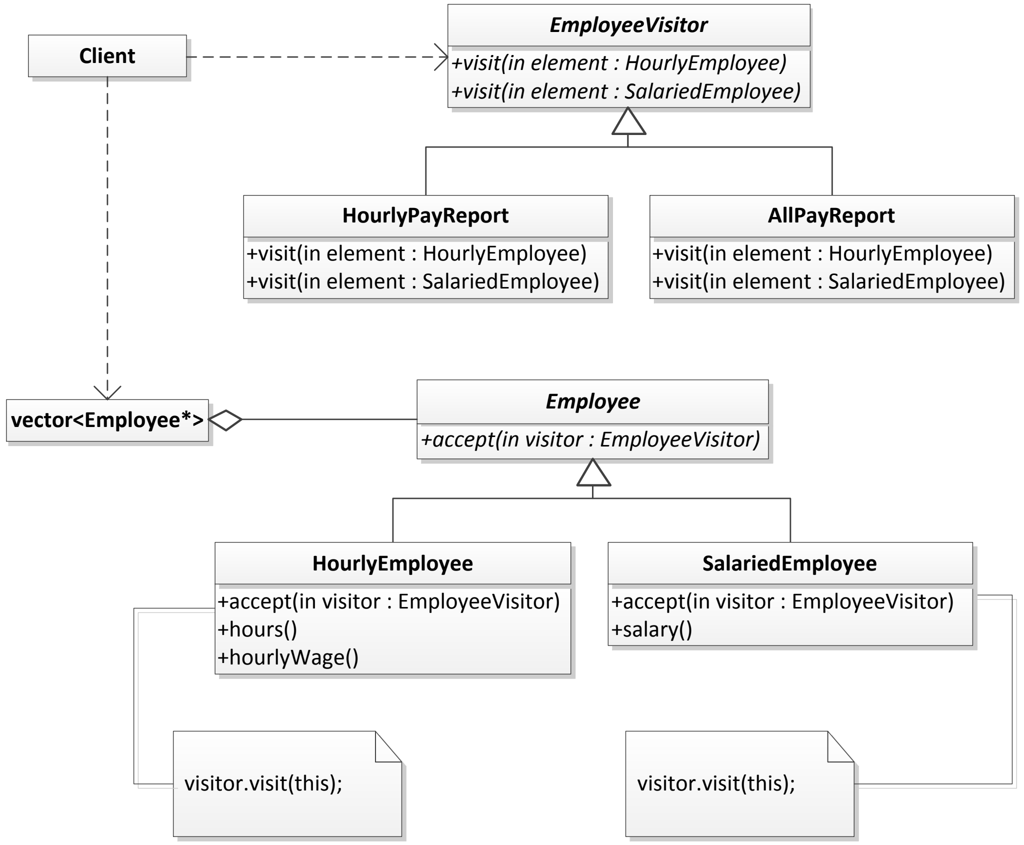
Behavioural Patterns: Visitor

- Applicability:
  - The classes defining the object structure rarely change (or cannot change), but you want to define new operations over the structure
  - For instance, we may be defining operations on third-party libraries classes to which we do not have the source code

Behavioural Patterns: Visitor


Client
Visitor
+visitConcreteElementA(in element : ConcreteElementA)
+visitConcreteElementB(in element : ConcreteElementB)
ConcreteVisitor1
+visitConcreteElementA(in element : ConcreteElementA)
+visitConcreteElementB(in element : ConcreteElementB)
ConcreteVisitor2
+visitConcreteElementA(in element : ConcreteElementA)
+visitConcreteElementB(in element : ConcreteElementB)
ObjectStructure
Element
+accept(in visitor : Visitor)
ConcreteElementA
+accept(in visitor : Visitor)
+operationA()
ConcreteElementB
+accept(in visitor : Visitor)
+operationB()
visitor.visitConcreteElementA(this)
visitor.visitConcreteElementB(this)

Behavioural Patterns: Visitor


Client
EmployeeVisitor
+visit(in element : HourlyEmployee)
+visit(in element : SalariedEmployee)
HourlyPayReport
+visit(in element : HourlyEmployee)
+visit(in element : SalariedEmployee)
AllPayReport
+visit(in element : HourlyEmployee)
+visit(in element : SalariedEmployee)
vector<Employee*>
Employee
+accept(in visitor : EmployeeVisitor)
HourlyEmployee
+accept(in visitor : EmployeeVisitor)
+hours()
+hourlyWage()
SalariedEmployee
+accept(in visitor : EmployeeVisitor)
+salary()
visitor.visit(this);
visitor.visit(this);

Behavioural Patterns: Visitor

Employee.h

```
class Employee
{
  public:
    Employee(const std::string& name) : _name(name) { }

    const std::string name() const
    {
      return this->_name;
    }

    virtual void accept(EmployeeVisitor*) = 0;

  protected:
    std::string _name;
};
```

Behavioural Patterns: Visitor

HourlyEmployee.cpp

```
void HourlyEmployee::accept(EmployeeVisitor* visitor)
{
  visitor->visit(this);
}
```

SalariedEmployee.cpp

```
void SalariedEmployee::accept(EmployeeVisitor* visitor)
{
  visitor->visit(this);
}
```

Behavioural Patterns: Visitor

EmployeeVisitor.h

```
class EmployeeVisitor
{
  public:
    virtual void visit(HourlyEmployee*) = 0;
    virtual void visit(SalariedEmployee*) = 0;
};
```

Behavioural Patterns: Visitor

HourlyPayReport.h

```
class HourlyPayReport : public EmployeeVisitor
{
  public:
    HourlyPayReport(std::ostream&);
    virtual void visit(HourlyEmployee*);
    virtual void visit(SalariedEmployee*);

  private:
    std::ostream& _out;
};
```

Behavioural Patterns: Visitor

HourlyPayReport.cpp

```
void HourlyPayReport::visit(HourlyEmployee* e)
{
  this->_out << setw(20) << e->name();
  this->_out << setw(10) << e->hours();
  this->_out << "$" << setw(9) << e->hourlyWage();
  this->_out << "$" << (e->hours() * e->hourlyWage()) << endl;
}

void HourlyPayReport::visit(SalariedEmployee* e)
{
  // Do nothing
}
```

Behavioural Patterns: Visitor

AllPayReport.cpp

```
void AllPayReport::visit(HourlyEmployee* e)
{
  this->_out << setw(20) << e->name();
  this->_out << setw(10) << "n/a";
  this->_out << "$" << setw(9) << e->hourlyWage() << endl;
}

void AllPayReport::visit(SalariedEmployee* e)
{
  this->_out << setw(20) << e->name();
  this->_out << "$" << setw(9) << e->salary();
  this->_out << setw(10) << "n/a" << endl;
}
```

Behavioural Patterns: Visitor

main.cpp

```
main()
{
  vector<Employee*> employees;

  employees.push_back(new HourlyEmployee("Joe User", 60, 25.75));
  employees.push_back(new HourlyEmployee("Jane Doe", 55, 31.25));
  employees.push_back(new SalariedEmployee("Bob Caygeon", 75000));
  employees.push_back(new SalariedEmployee("Eve Adams", 72000));

  HourlyPayReport rpt1(cout);

  for (vector<Employee*>::iterator it = employees.begin(); it != employees.end(); ++it)
    (*it)->accept(&rpt1);    // Why not call rpt1.visit(*it)?  The visit() method requires a pointer
                             // to an instance of a concrete subclass, not the abstract parent class,
                             // as each concrete subclass is potentially treated differently.
  cout << endl;

  AllPayReport rpt2(cout);

  for (vector<Employee*>::iterator it = employees.begin(); it != employees.end(); ++it)
    (*it)->accept(&rpt2);
}
```

Behavioural Patterns: Visitor

Output

```
Hourly Employee Pay Report
Name                          Hours          Wages          Pay
========================================================================
Joe User                      60             $25.75  $1545
Jane Doe                      55             $31.25  $1718.75
Employee Pay Report
Name                          Salary         Wage
========================================================================
Joe User                      n/a            $25.75
Jane Doe                      n/a            $31.25
Bob Caygeon                   $75000  n/a
Eve Adams                     $72000  n/a
```

Behavioural Patterns: Visitor

- Consequences:
  - Visitor makes adding new operations easy
  - A visitor gathers related operations and separated unrelated ones
  - Accumulating state
  - Adding new ConcreteElement classes is hard

Behavioural Design Patterns

- State
- Strategy
- Observer
- Command
- Visitor