Design Pattern

Capitolo 10, sezione 10.4 esercizi 10.3 e altri

Singleton 10.4 (pag 291)

- A single instance of the class
- In Java
  - private constructor
  - static member

```
class A{
   private A(){...}
   public static A instance = new A();

}
```

Visitor Pattern es. 10.3

Synopsis

- 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.

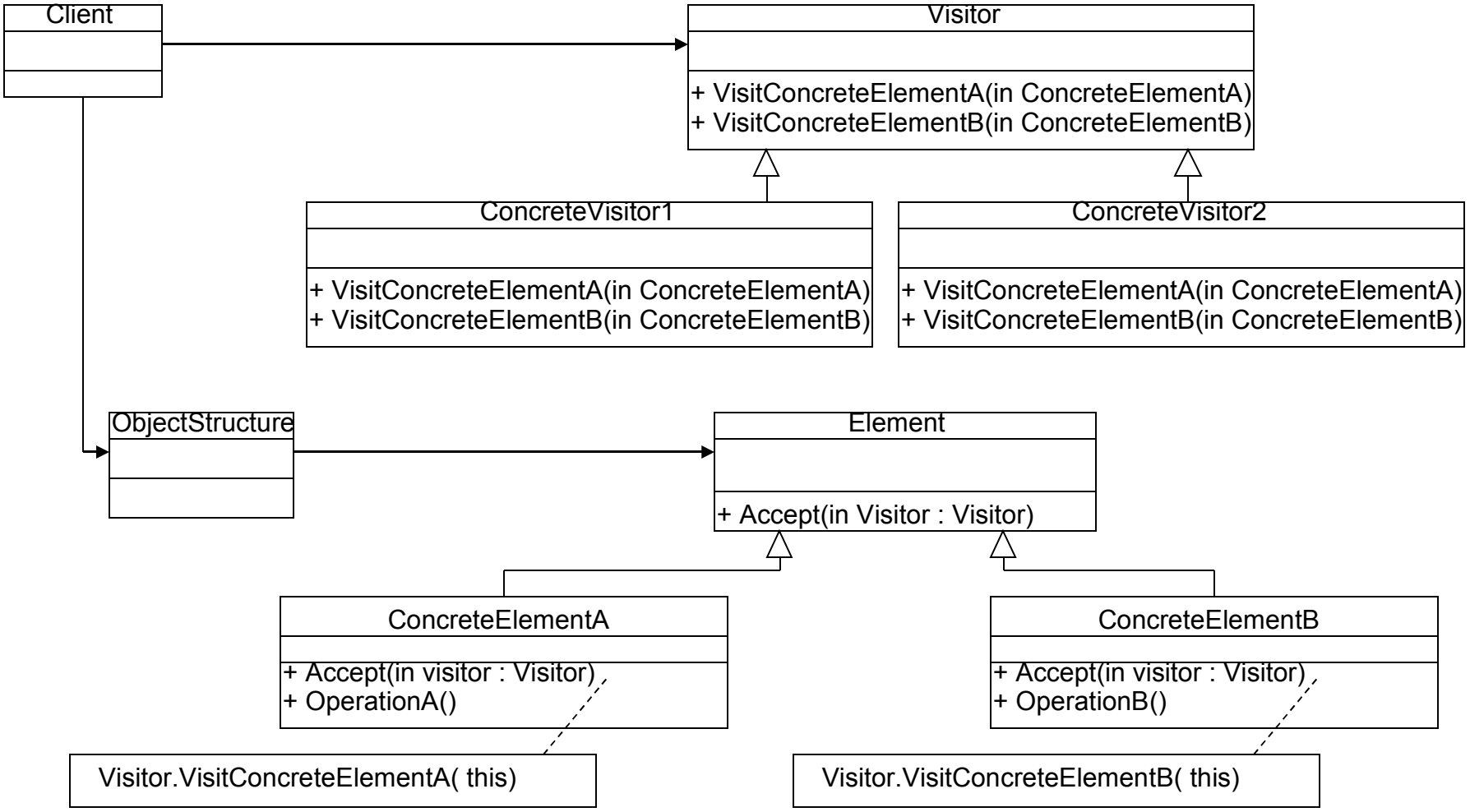
Visitor Pattern

- Problem
  - Operations on collections of objects may not apply to all objects, or apply differently to different objects
- Context
  - Object interfaces are fixed and diverse
  - Need to allow new operations, without polluting” their classes with these operations.
- Solution
  - Represent operations to be performed as visitors, with the interface of every visitor representing the different kinds of objects

Context

- You should use the Visitor pattern when:
  - An object contains many classes of objects with differing interfaces.
  - Many distinct and unrelated operations need to be performed on an object structure, and you want to avoid “polluting” their classes with these operations.
  - The classes defining the object structure rarely change, but you often want to define new operations over the structure.

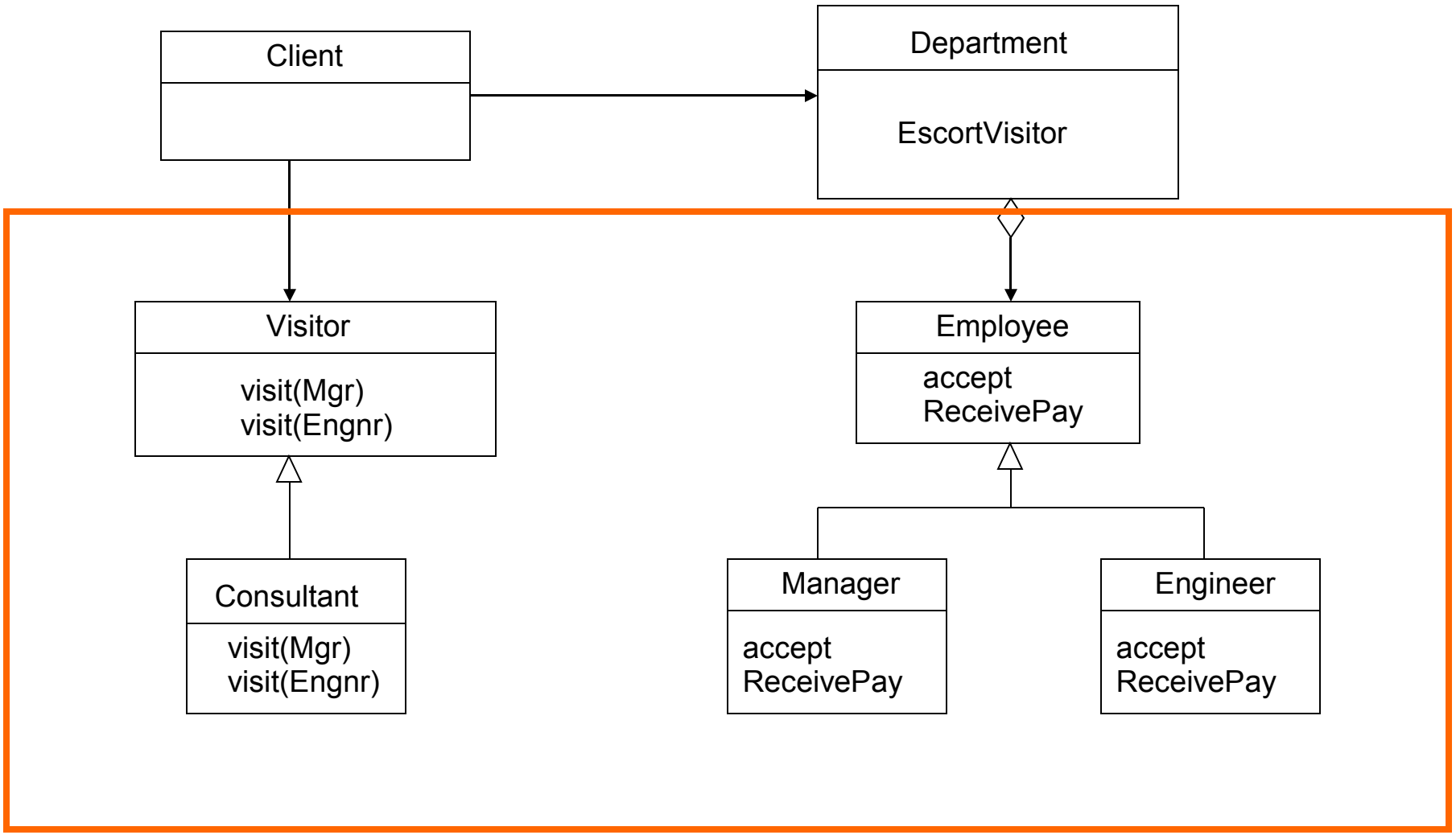
Visitor Pattern Diagram



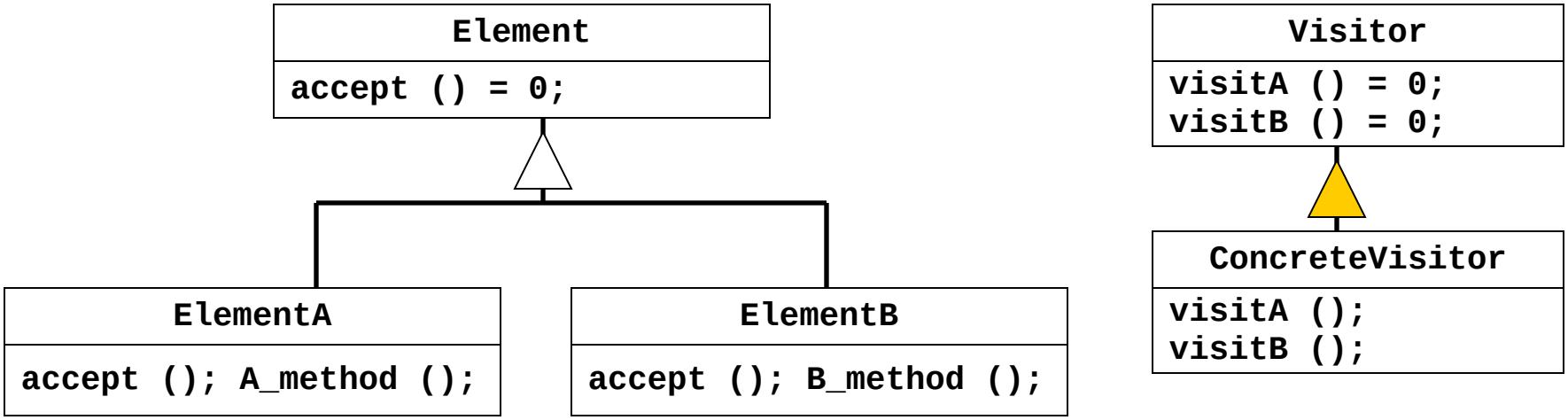
Visitor Example

- Imagine you have a department, with two type of employee’s managers and engineer’s.
- You want to take a survey of your employee’s to see how you can make there workplace better.
- The best way to do this is to have a consultant (visitor) come in and conduct the survey rather then having the two different types of do it themselves.

Visitor Example



Visitor Structure



- Different elements have different concrete interfaces
- Element abstract base class adds accept interface
- Double hand-shake between concrete element and visitor allows visitor to call the appropriate concrete element method

lato Visitable

- Le classi della gerarchia devono essere visitabili:

```
interface Visitable {
   void accept(Visitor v);
}
```

- ogni classe originale deve implementare visitable (un metodo accept):

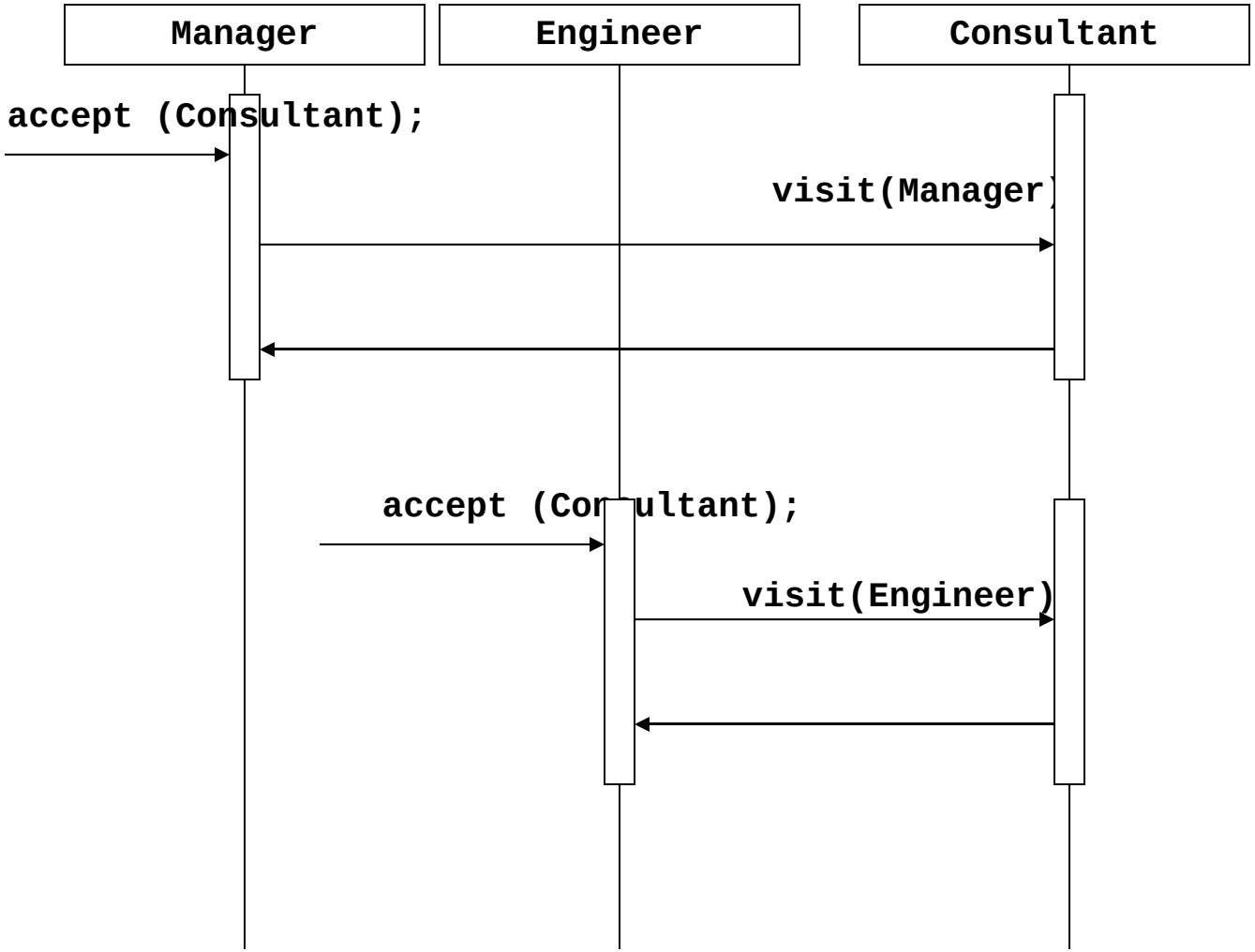
```
class Employee implements Visitable{
...
    accept(Visitor v){ v.visit(this);}
}
```

lato visitor

- consultant deve essere un visitor

```
public interface Visitor {
    public void visit(Manager m);
    public void visit(Engineer m);
}
public class Consultant implements Visitor{
    public void visit(Engineer m){
        System.out.println("consulting engineer");
     }
    public void visit(Manager m){
        System.out.println("consulting manager");
     }
```

Visitor Interactions



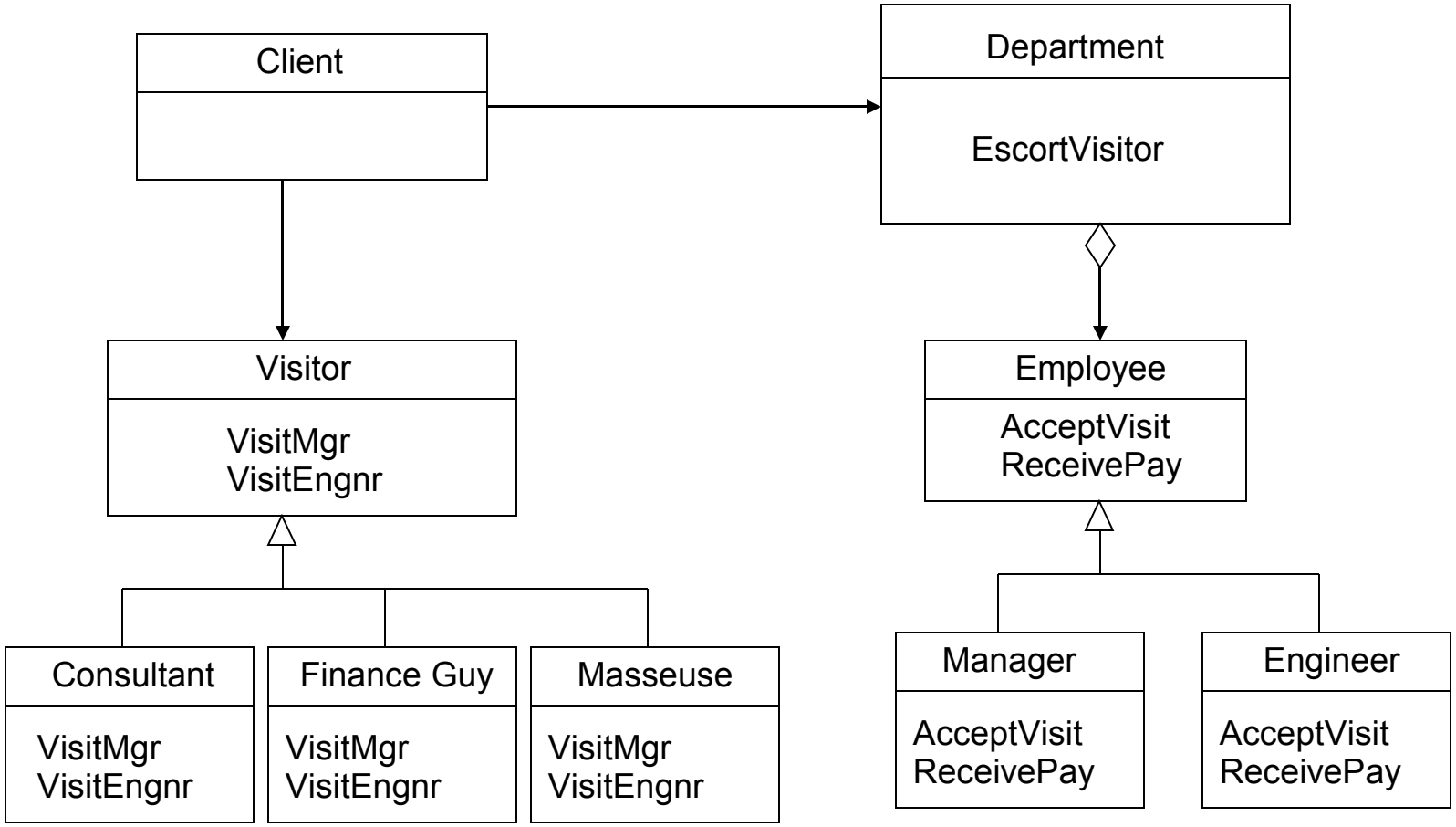
Visitor Example

- After the consultant surveys the employee's, the managers realizes that the employee's are both stressed and having financial troubles.

- So the managers then bring in a masseuse to help relieve stress at the office. They also have the finance guy go around and talk to the employee's to help them out.

BASTA AGGIUNGERE UN ALTRO VISITOR

Visitor Example



Implementation

- Esercizio 10.3 con Expression

Consequences

- Positive
  - Visitor makes adding new operations easy, simply add a new visitor that implements that operation.
  - Visitor gathers related operations and separates unrelated ones.

Consequences

- Negative
  - Visitor is not good for the situation where "visited" classes are not stable. Every time a new Composite hierarchy derived class is added, every Visitor derived class must be amended
  - Often encapsulation is broken because the element class is forced to provide public operations that access internal state.
  - If using an existing system changes will be required to existing code.

Related Patterns

- Iterator
  - The iterator pattern is an alternative to the Visitor pattern when the object structure to be navigated has a linear structure.
- Composite
  - The visitor pattern is often used with object structures that are organized according to the composite pattern.

Come fare restituire un valore da un visitor

Normalmente il visitor non restituisce niente. E se devo calcolare qualcosa: come fare? due alternative

2. modifica di visit
   - definire i metodi visit che restutiscano un valore
   - ad esempio restituiscano un Object
     - `Object visit(X…);`
   - poi faccio il cast sapendo cosa effettivamente restituisce
3. aggiungere un campo e un metodo
   - campo result, che viene settato alla fine della visita
   - getResult che restituisce il risultato della visita

visitor e generics

- L'alternativa è dichiarare il Visitor generico rispetto il tipo che restituisce:

```
public interface Visitor <T> {
    public T visit(Manager m);
    public T visit(Engineer e);
}
// if the visitor returns a String
public class Consultant implements Visitor<String>{

    public String visit(Engineer m){
        return "consulting engineer");    }

....
```

Generic Visitable

- And a generic Visitable with a generic method

```
interface Visitable{
 public <T> T accept(Visitor<T> ask);
}
class Manager implements Visitable{
 public <T> T accept(Visitor<T> ask){
        return ask.visit(this);
 }
} ...
```