

A photograph of a red squirrel climbing a tree trunk. The squirrel is facing away from the camera, its reddish-brown fur contrasting with the dark, textured bark of the tree. It is gripping the trunk with its front paws and a small branch. The background is filled with green foliage and other trees, creating a natural, woodland atmosphere.

## M8(a)- (More on) Design Patterns

---

Jin L.C. Guo

*Image source: <https://www.goodfreephotos.com/albums/animals/mammals/red-squirrel-climbing-up-a-tree.jpg>*

# Objective

- Be able to use the Visitor Design Pattern effectively;
- Be able to understand the major categories of design patterns
- Be able to determine when to used different design patterns effectively

# Objective

- Be able to use the Visitor Design Pattern effectively;
- Be able to understand the major categories of design patterns
- Be able to determine when to used different design patterns effectively



## University

### Faculty of Art

English

Philosophy

Sociology

...

### Faculty of Science

CS

Biology

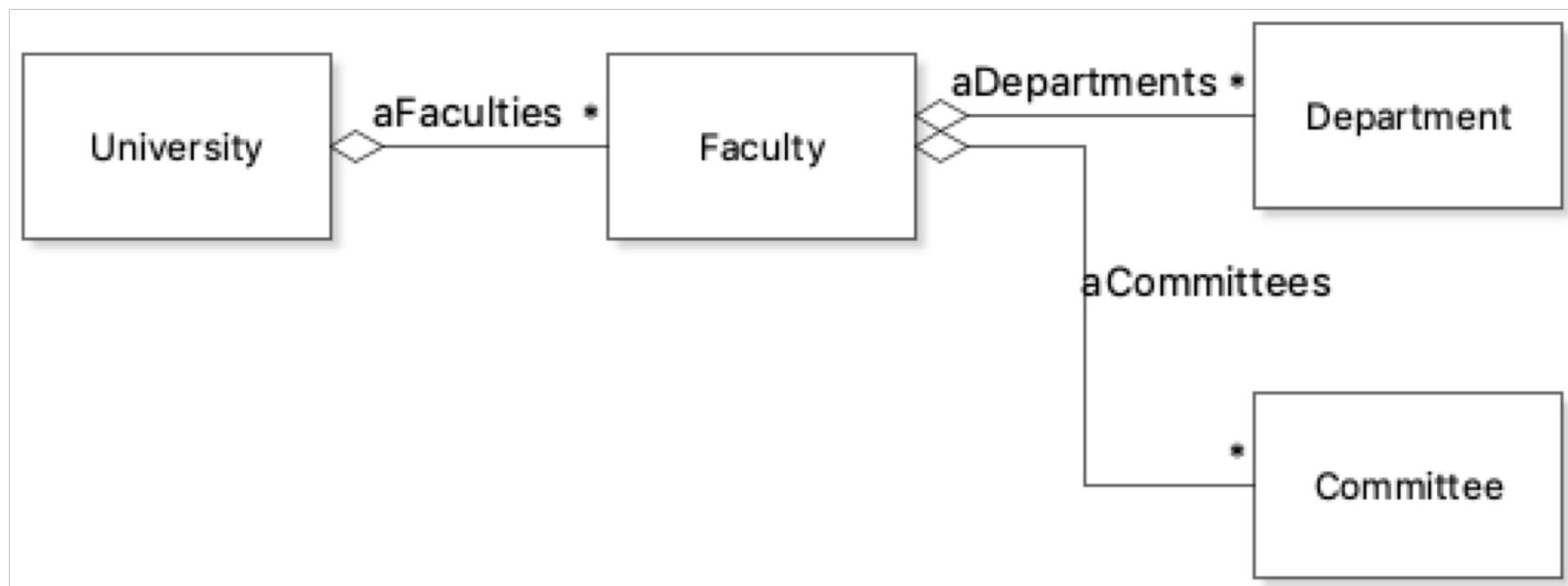
Physics

...

Academic Committee

Scholarship Committee

Students Committee



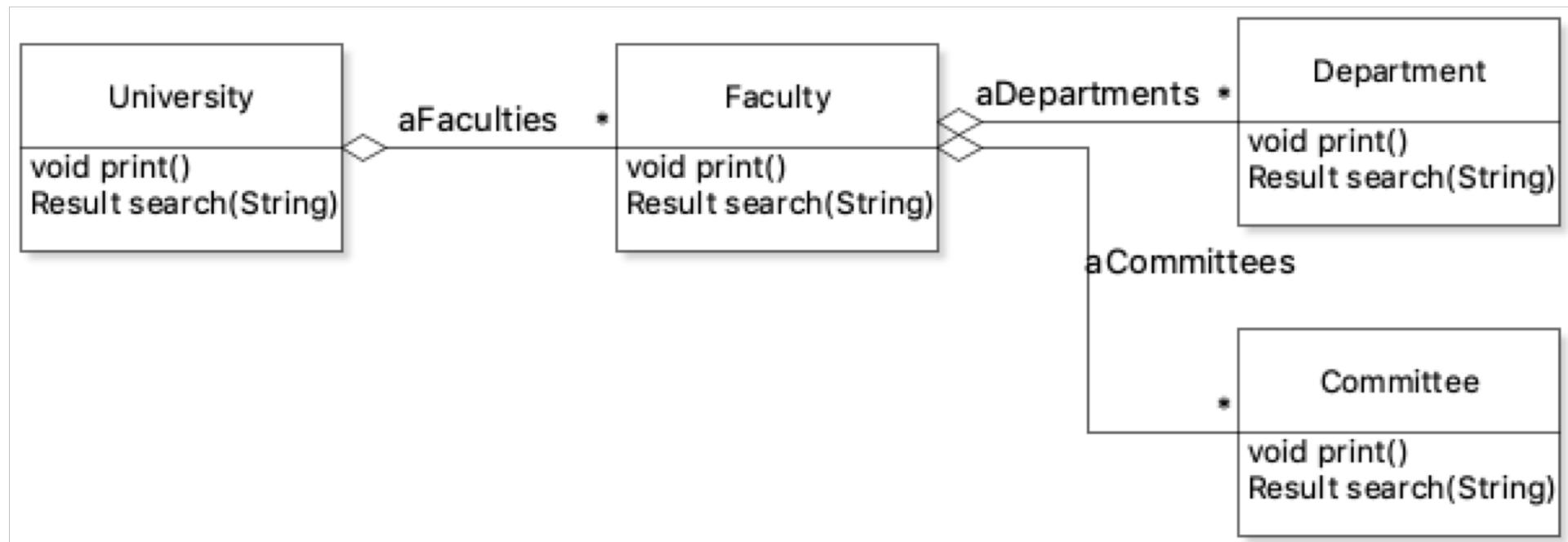
# Add functionalities to aggregate

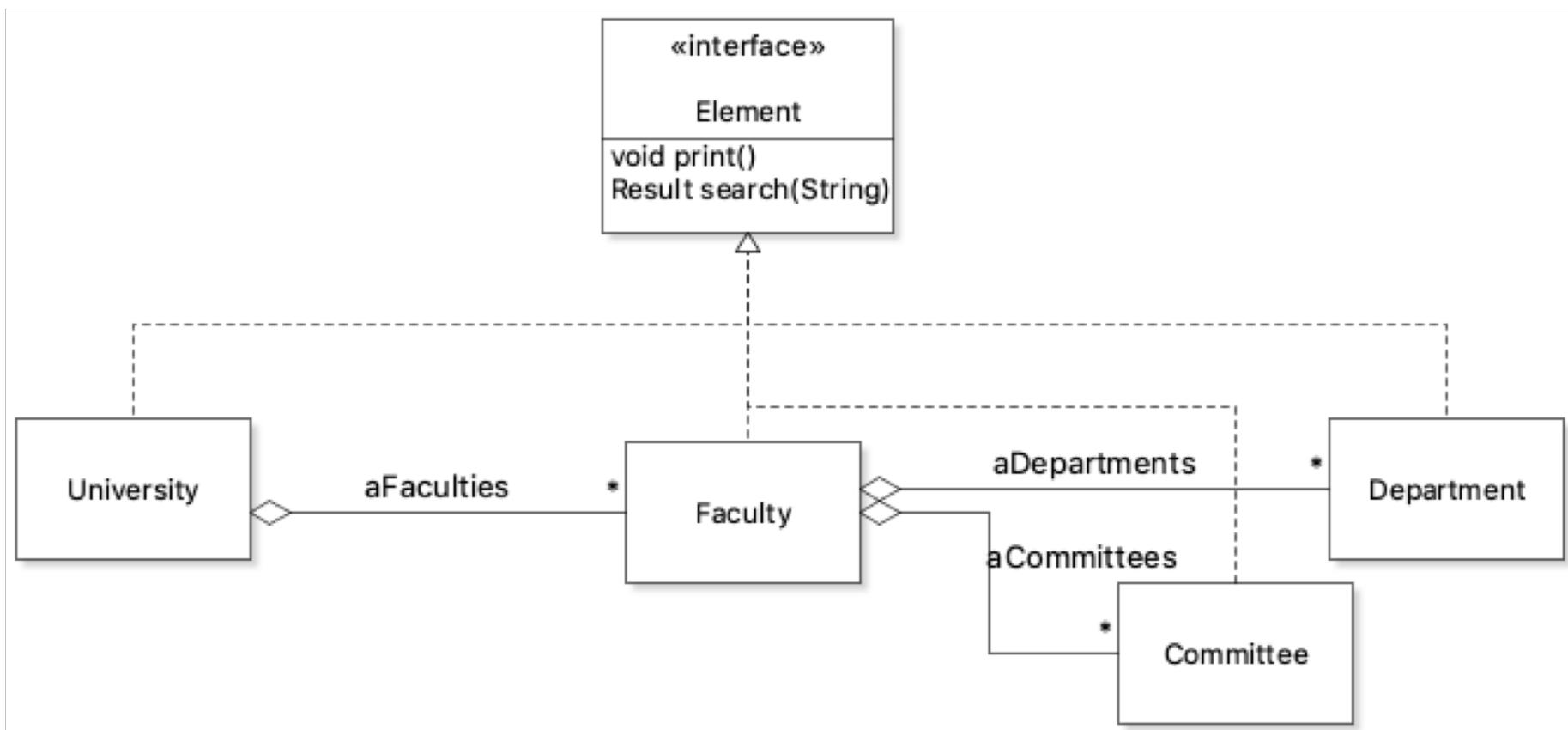
Print annual report for every organizations in university

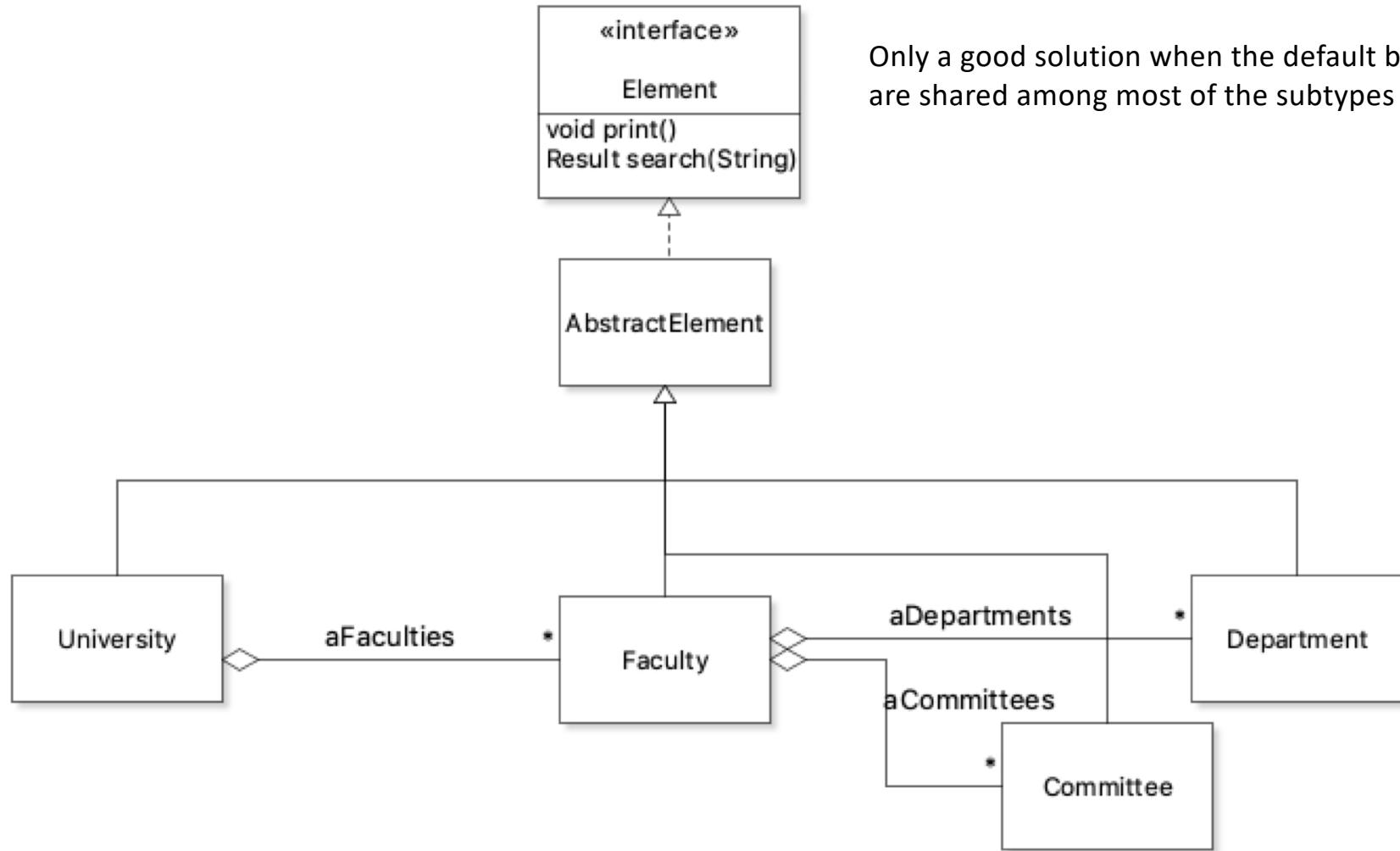
Search if one person belongs to any organization in university

... ...

All requires traverse all the elements in University





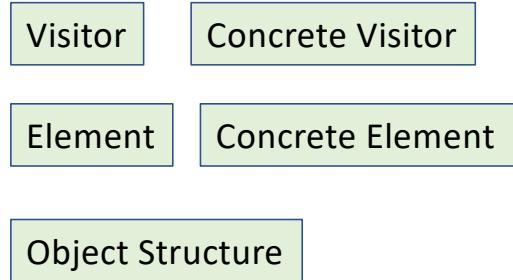


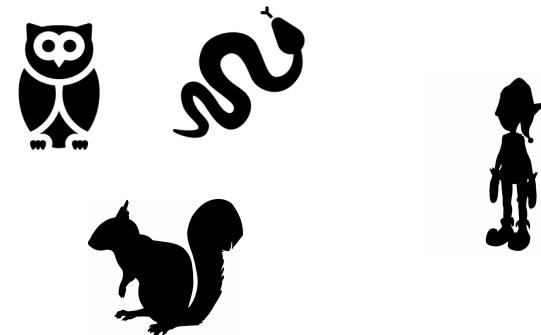
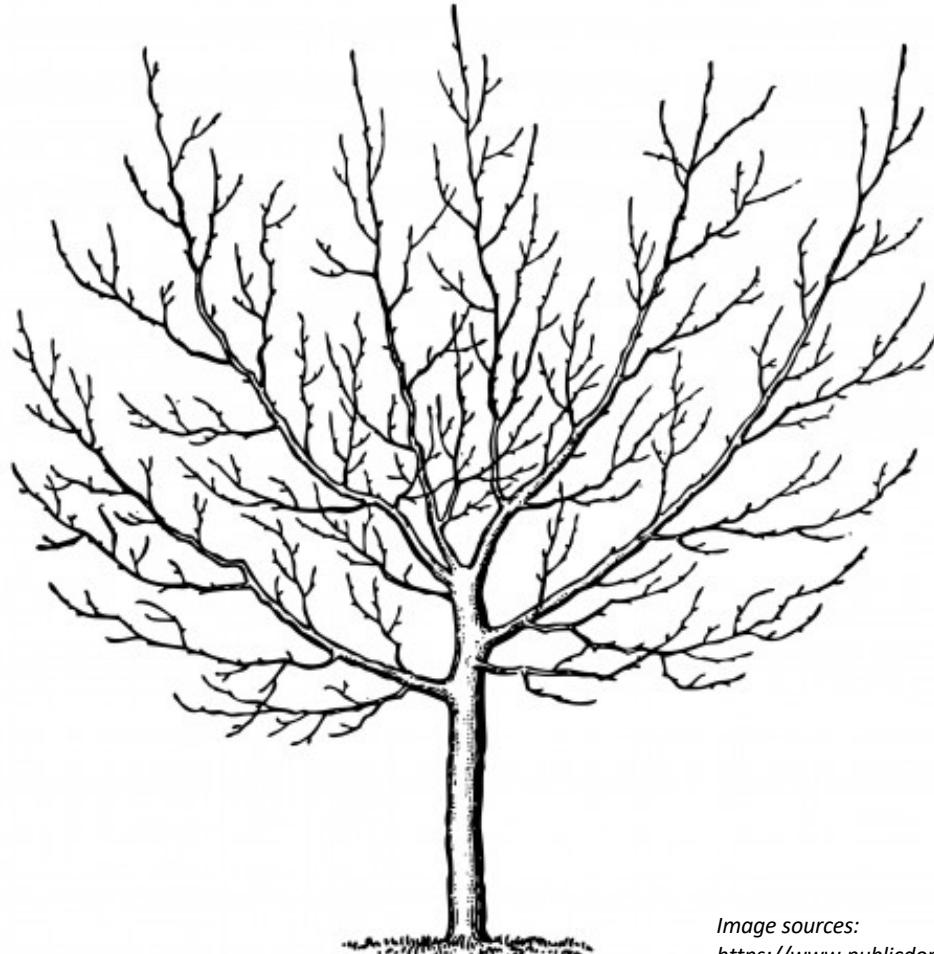
# Visitor

- Intent:

*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.*

- Participants:



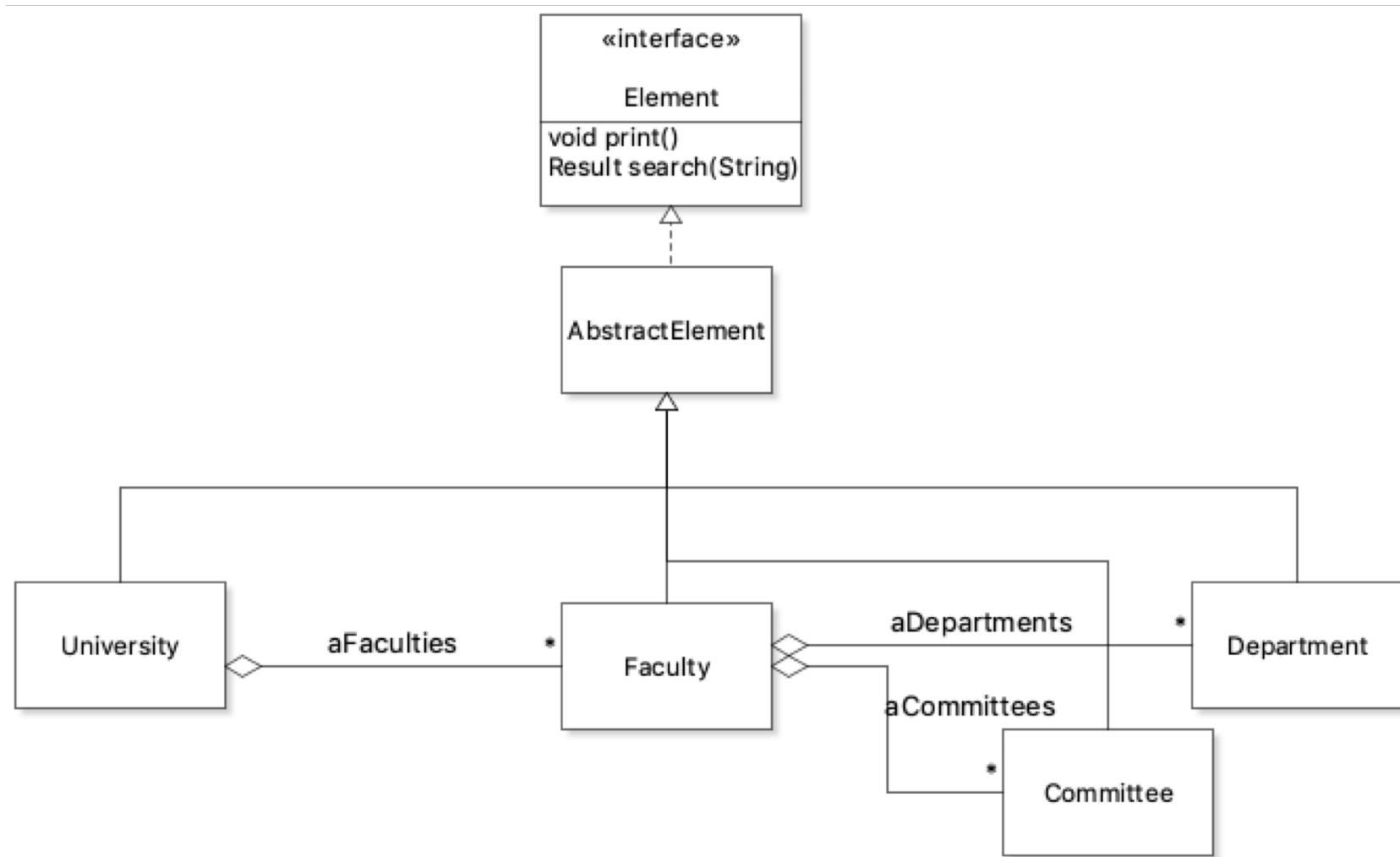


*Image sources:*

<https://www.publicdomainpictures.net/pictures/100000/nahled/tree-1409250600Al7.jpg>

<https://www.publicdomainpictures.net/pictures/140000/nahled/black-elf.jpg>

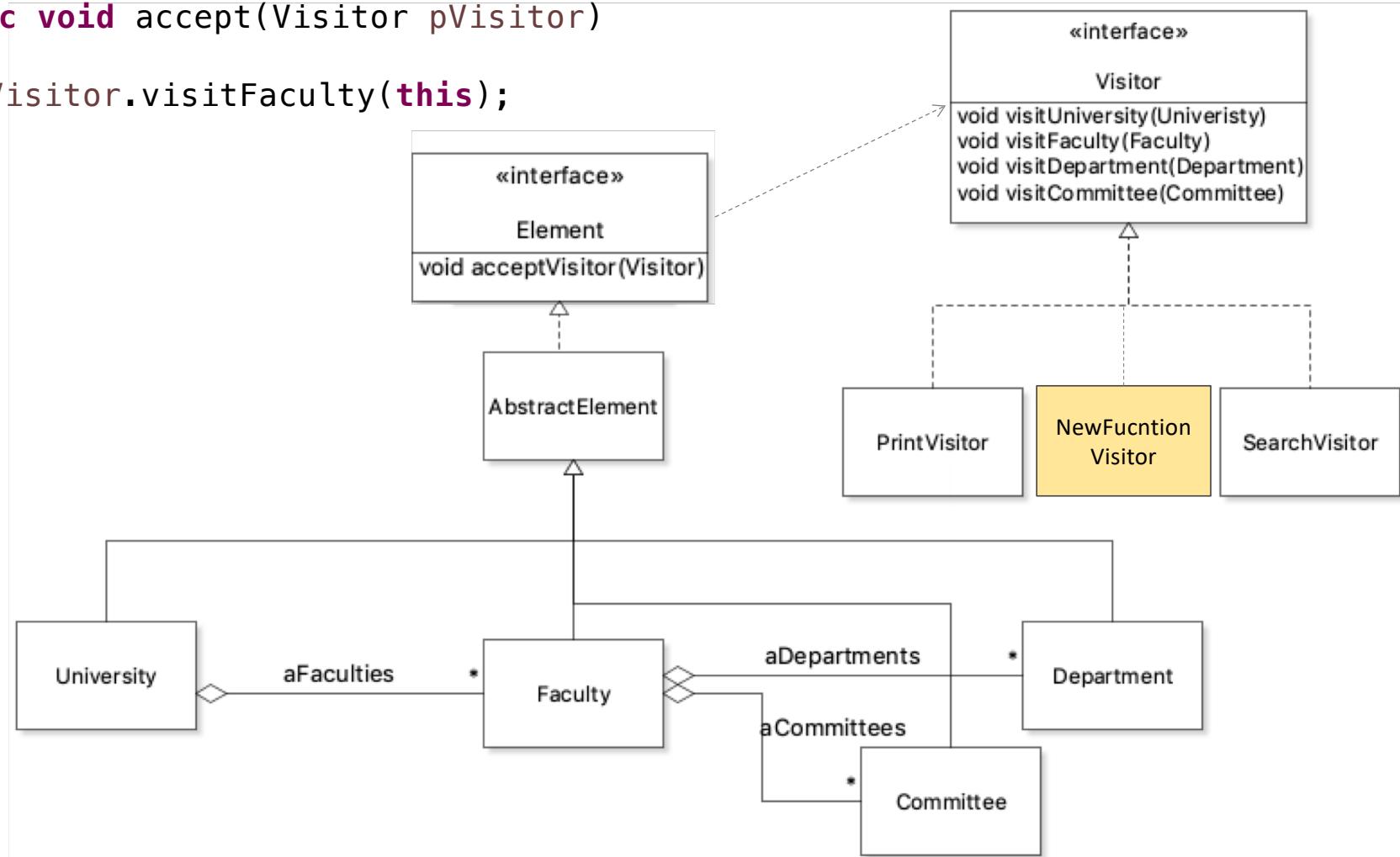
<https://www.publicdomainpictures.net/pictures/190000/nahled/squirrel-silhouette-1469799208bea.jpg>

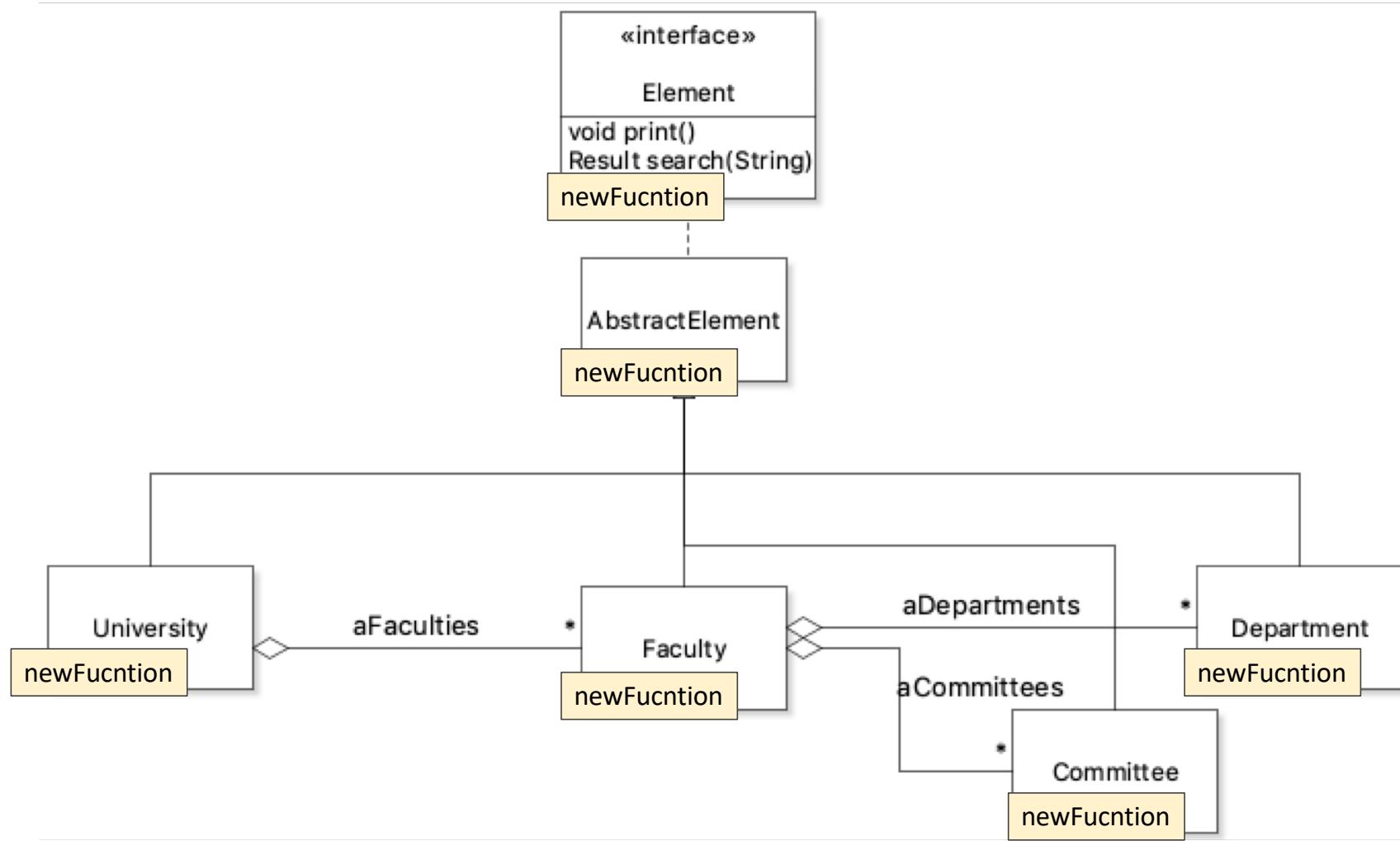


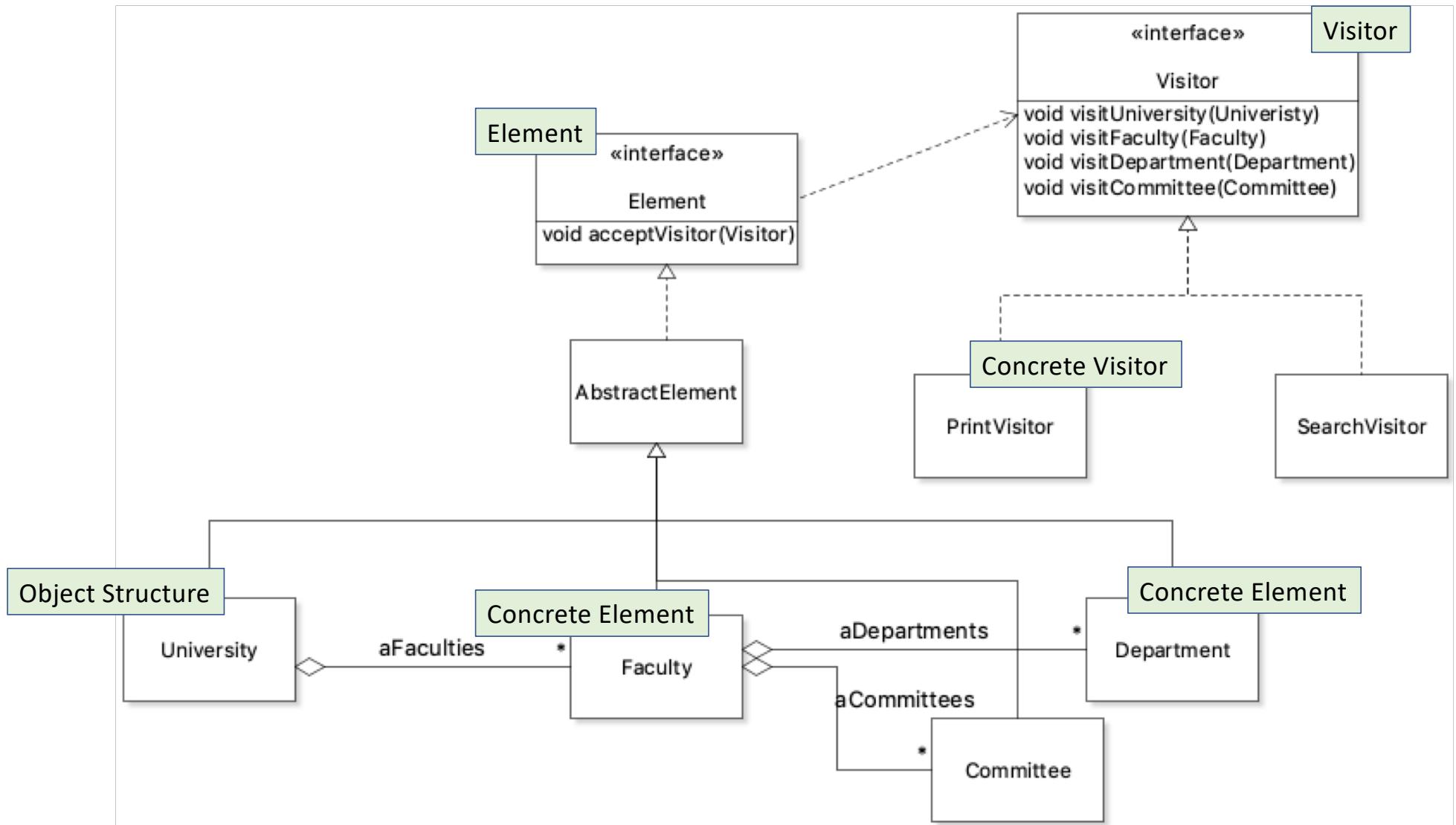
```

@Override
public void accept(Visitor pVisitor)
{
    pVisitor.visitFaculty(this);
}

```







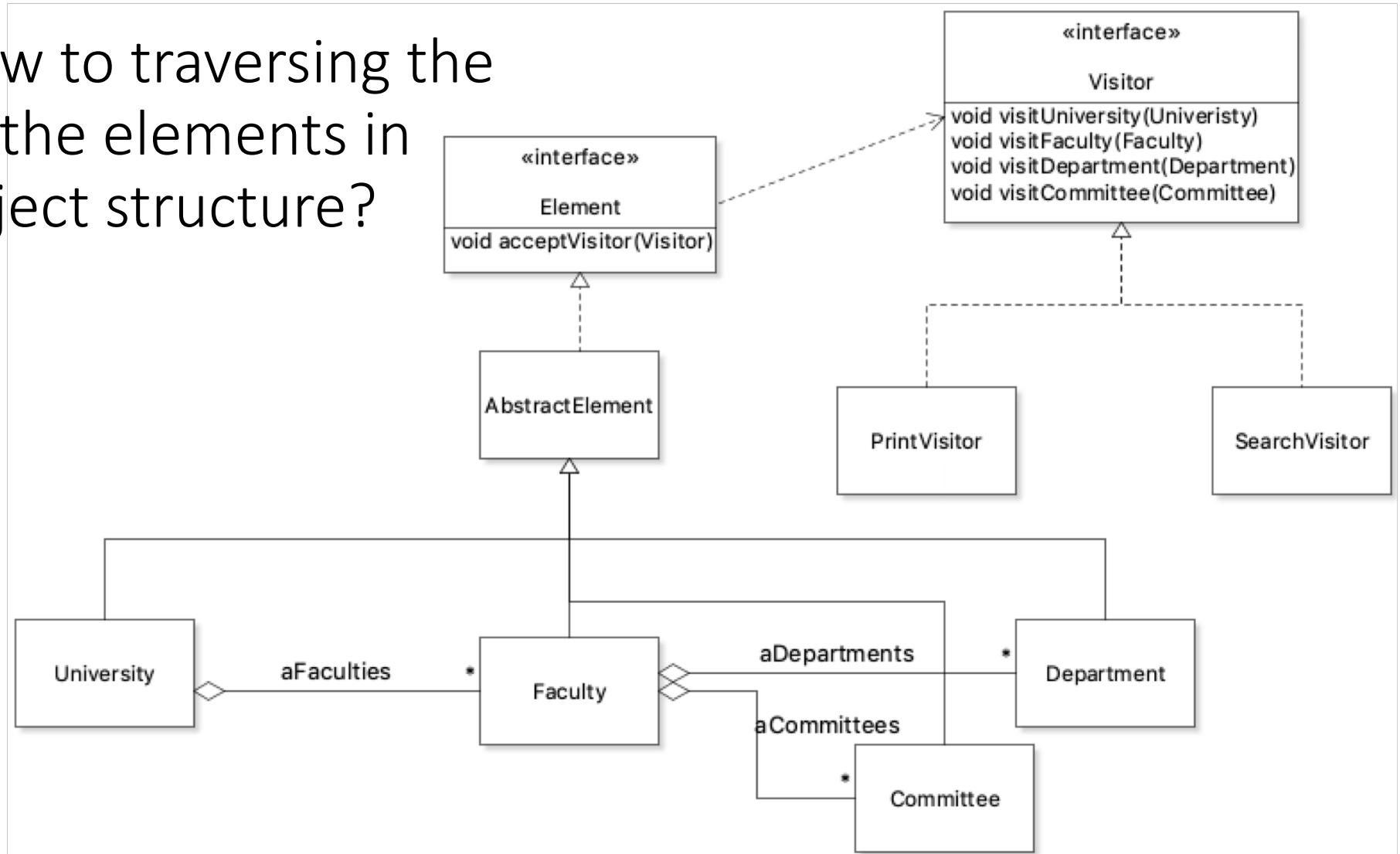
## Activity 1:

- Consequences of applying Visitor pattern.

# True or False?

- A visitor gathers related operations and separates unrelated ones.
- Adding new `ConcreteElement` classes becomes easy.
- `ConcreteElement` interface has to provide public operations that access an element's internal state, so it may compromise its encapsulation.
- Visitor pattern makes it relatively easy to accumulate state since single visitor can visit each element in the object structure.

# How to traversing the all the elements in object structure?



## Method1:

```
class University extends AbstractElement
{
    private final List<Faculty> aFaculties = new ArrayList<Faculty>();
    public University(String pName) { super(pName); }
    public void addFaculty(Faculty pFaculty) { aFaculties.add(pFaculty); }

    @Override
    public void accept(Visitor pVisitor)
    {
        pVisitor.visitUniversity(this);

        for( Faculty f : aFaculties )
        {
            f.accept(pVisitor);
        }
    }
}
```

← Traverse in the object structure.

## Method1:

```
public class PrintVisitor implements Visitor
{
    @Override
    public void visitUniversity(University pUniversity)
    {
        // Printing operation for University
    }

    @Override
    public void visitFaculty(Faculty pFaculty)
    {
        // Printing operation for Faculty
    }
}
```

## Method2:

```
class University extends AbstractElement
{
    private final List<Faculty> aFaculties = new ArrayList<Faculty>();
    public University(String pName) { super(pName); }
    public void addFaculty(Faculty pFaculty) { aFaculties.add(pFaculty); }
    public Iterator<Faculty> getFaculties() { return aFaculties.iterator(); }
```

↑ But provide a traversal mechanism to its elements

```
@Override
public void accept(Visitor pVisitor)
{
    pVisitor.visitUniversity(this);
}
```

← Not in the object structure.

## Method2:

```
public class PrintVisitor implements Visitor
{
    @Override
    public void visitUniversity(University pUniversity)
    {
        // Printing operation for University
        for( Iterator<Faculty> i = pUniversity.getFaculties();
            i.hasNext(); )
        {
            i.next().accept(this);
        }
    }
}
```

Traverse in the visitor→

```
for( Iterator<Faculty> i = pUniversity.getFaculties();
    i.hasNext(); )
{
    i.next().accept(this);
}
```

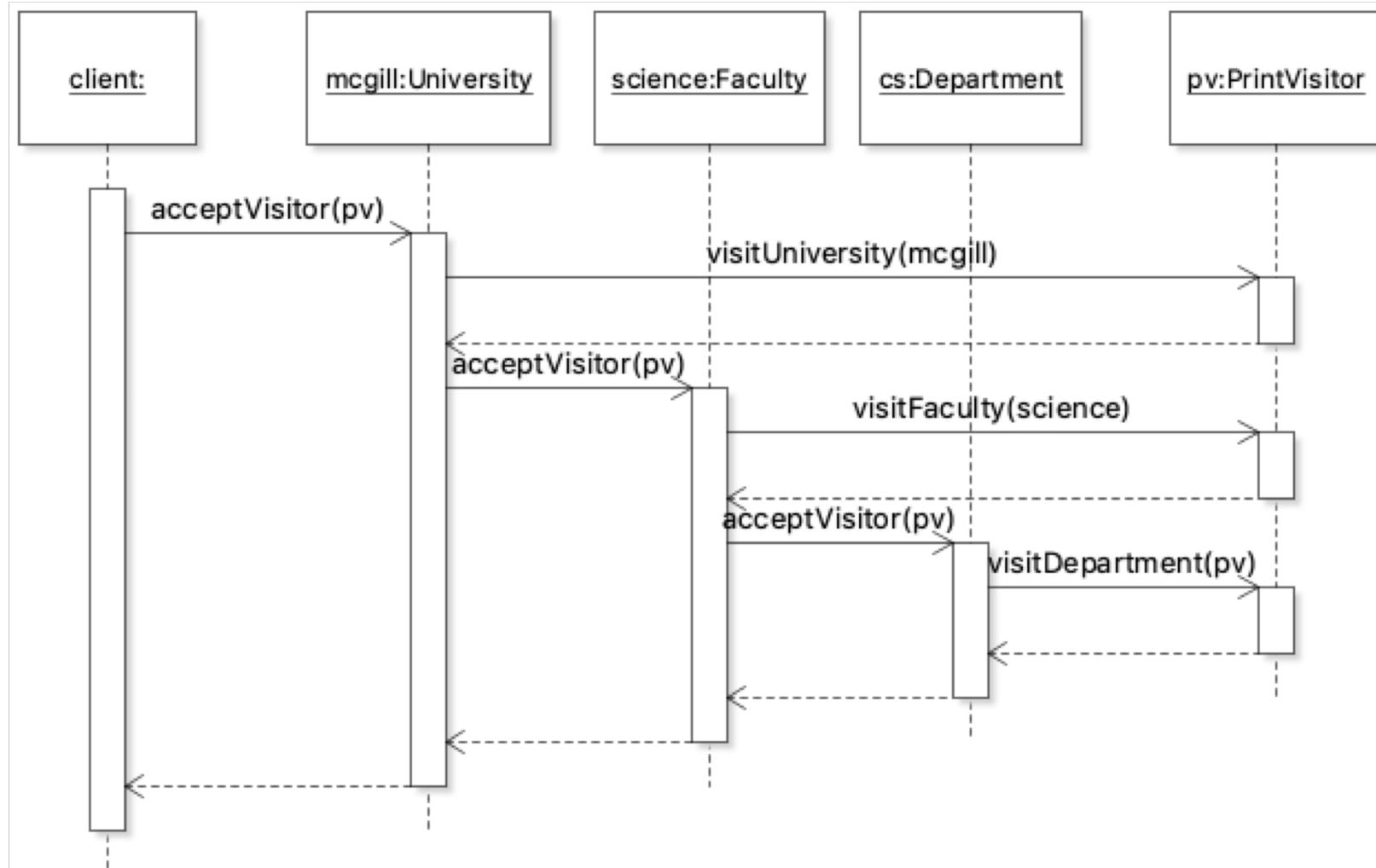
## Activity2: Sequence Diagram for Visitor pattern

```
University mcGill = new University("McGill");
Faculty science = new Faculty("Science");
Department cs = new Department("Computer Science");

mcGill.addFaculty(science);
science.addDepartment(cs);

Visitor pv = new PrintVisitor();
mcGill.acceptVisitor(pv);
```

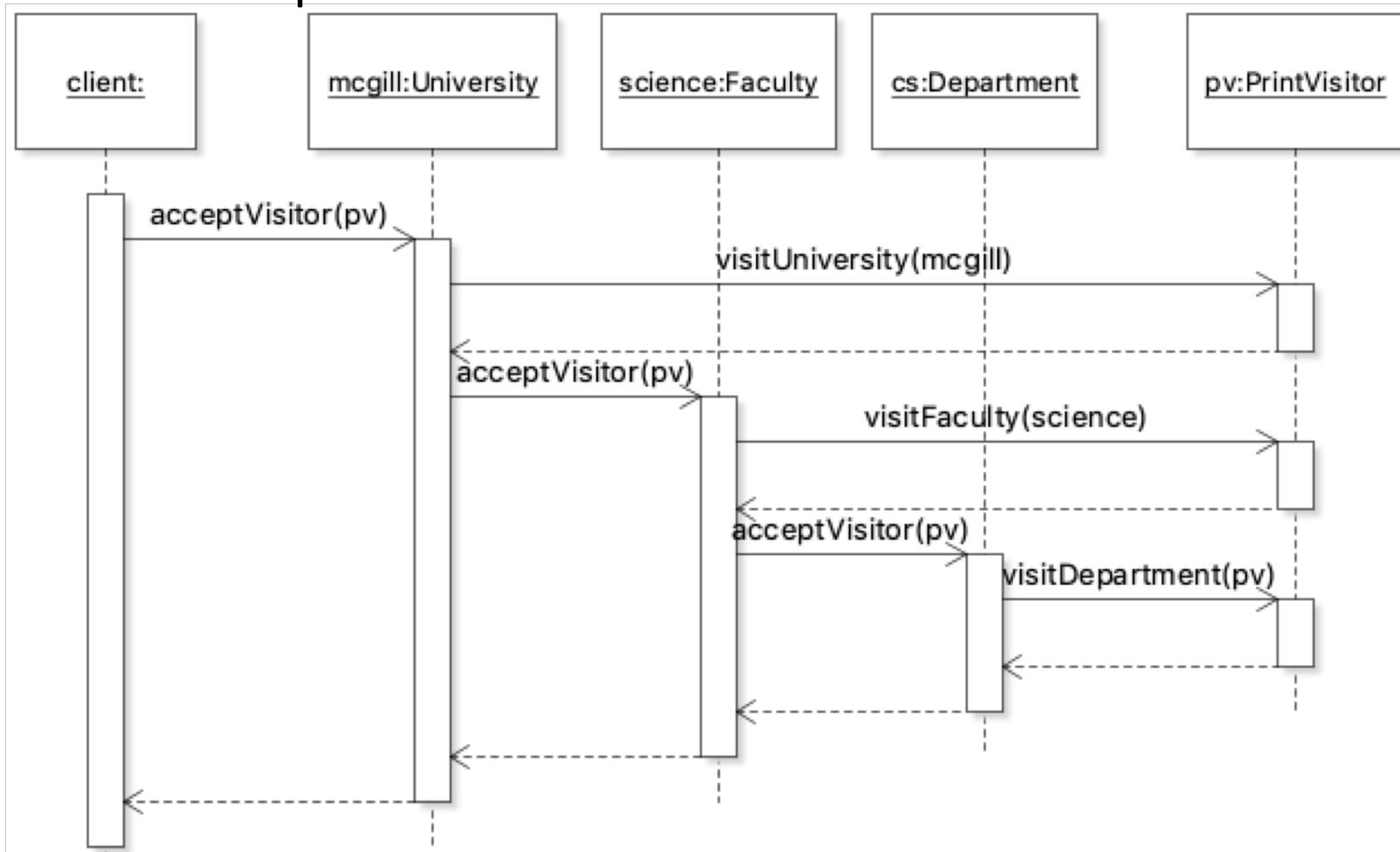
## Traverse in the object structure



Sequence diagram for traversal in Visitor?

# Double Dispatch

*The actual operation is dispatched twice before getting executed.*



# Demo on McGill Visitor

- Double Dispatch
- Different Traversal Strategy
- Composite Pattern with Visitor
- How Nullability is handled.