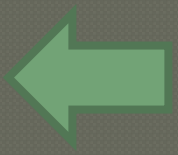
Design Patterns

- The Strategy Pattern
- The Factory Method
- Generics
- The Abstract Factory Pattern
- The State Pattern
- The Observer Pattern
- The Adapter Pattern
- The Composite Pattern
- The Iterator Pattern
- The Builder Pattern
- Fallen Patterns
  - The Singleton Pattern
  - **The Visitor Pattern**
- Command Pattern



Visitors

A way of "conveniently" iterating through a collection

Visitor as a method called "visit" it runs for each element

Each element has an "accept" method which takes the visitor

Preserves composition

Example

- We can apply this pattern to our HTML DOM from last class

```
interface TagVisitor {
  void visitTag( String name, String id, … );
}

class Tag {
  private String name;
  private String id;
  private Tag[] children;
  //…
  public void accept(TagVisitor v) {
    v.visitTag( name, id, … );
    foreach( Tag child : children ) {
      child.accept( v );
  }
}
```

Example

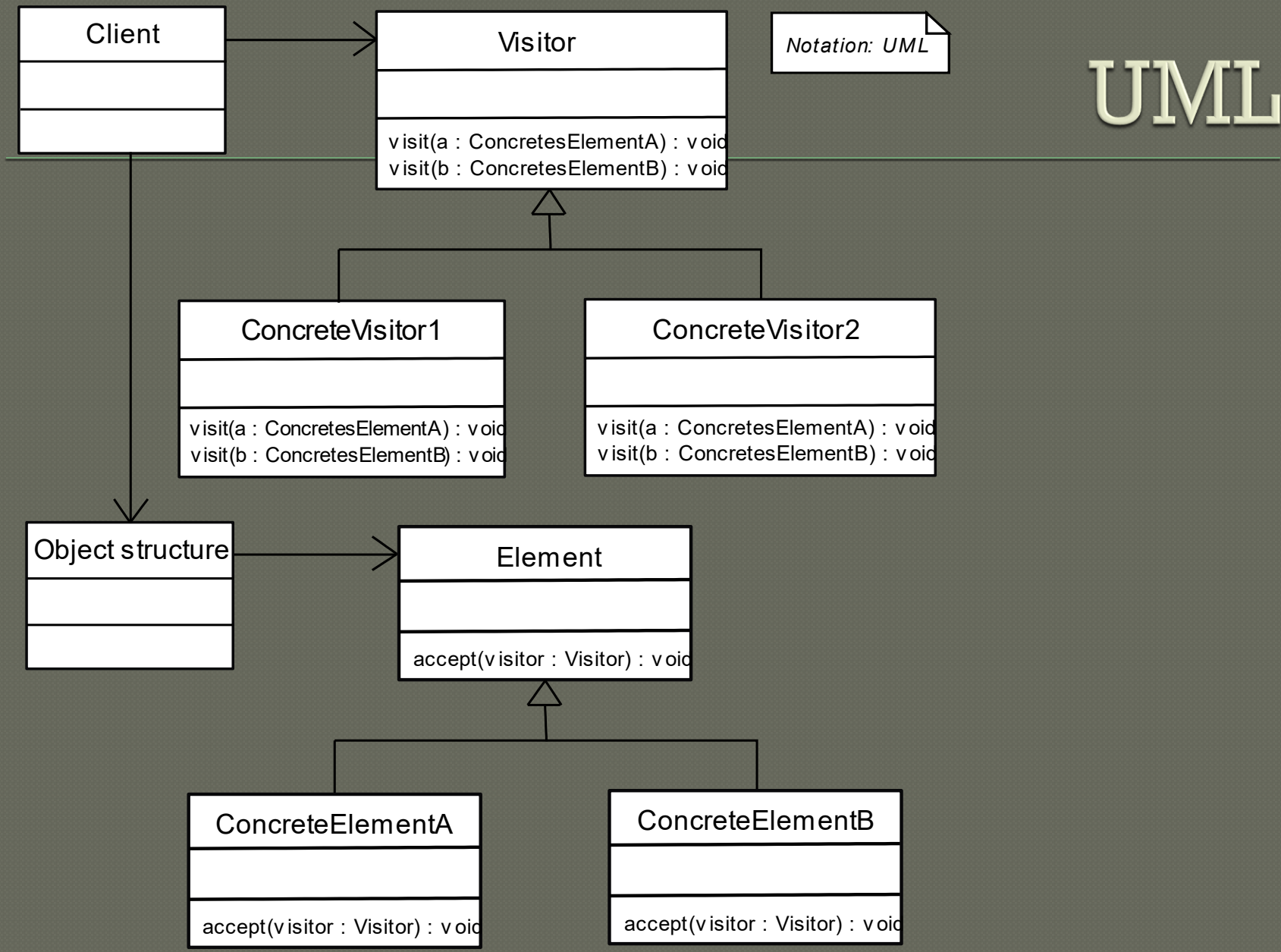
```
class TagPrinter implements TagVisitor {

  @Override
  void visitTag( String name, String id, … ) {
    System.out.println( name );
    //…
  }
}
```

Extending the Example

- We can have multiple kinds of visitors inherit from TagVisitor
- We can have multiple kinds of elements to iterate over
  - TagVisitor has a visit method for each kind of thing it can visit

UML


Client
Visitor
visit(a : ConcretesElementA) : voic
visit(b : ConcretesElementB) : voic
Notation: UML
ConcreteVisitor1
visit(a : ConcretesElementA) : voic
visit(b : ConcretesElementB) : voic
ConcreteVisitor2
visit(a : ConcretesElementA) : voic
visit(b : ConcretesElementB) : voic
Object structure
Element
accept(visitor : Visitor) : voic
ConcreteElementA
accept(visitor : Visitor) : voic
ConcreteElementB
accept(visitor : Visitor) : voic

Differences Between Iterator and Visitor

Iterators

- Lazy
- Stateful
- Don't require special support (main benefit)
- Not encapsulated
- Don't usually mutate
- Can "wrap" other iterators

Visitors

- Eager
- Usually Stateless
- Require the class being visited to support visitor
- Heavily encapsulated
- Often mutate
- Can't be composed easily

Visitors: Mostly Obsolete

- Only for extreme forms of encapsulation: rarely worth it
- Forced to create a separate method for each kind of thing to visit (bad polymorphism)
- Iterators let you make visitors without a fixed interface
- Functional programming is the nail in the coffin:
  - ‘map’/’reduce’ replaces most visitors conveniently
  - ‘foreach’ replaces *all* visitors