Partial Solution Part II
Review: Visitor Pattern

![Diagram of Visitor Pattern]

- **Visitor**
  - `VisitConcreteElementA(ConcreteElementA)`
  - `VisitConcreteElementB(ConcreteElementB)`

- **ConcreteVisitor1**
  - `VisitConcreteElementA(ConcreteElementA)`
  - `VisitConcreteElementB(ConcreteElementB)`

- **ConcreteVisitor2**
  - `VisitConcreteElementA(ConcreteElementA)`
  - `VisitConcreteElementB(ConcreteElementB)`

- **ObjectStructure**
  - **Element**
    - `Accept(Visitor)`

- **ConcreteElementA**
  - `Accept(Visitor v)`
  - `OperationA()`
  - `v->VisitConcreteElementA(this)`

- **ConcreteElementB**
  - `Accept(Visitor v)`
  - `OperationB()`
  - `v->VisitConcreteElementB(this)`
Visitor (con’t)

• The Visitor substructure represents the hierarchy of the actions/operations
• The Element substructure represents the hierarchy of the objects
• Creating $n$ by $m$ methods instead of creating $n$ by $m$ class hierarchy since each object needs its own action and each action needs to be defined for every object
  – helps to avoid creating overwhelmingly many of classes
• Particularly useful when the object structure is of Composite type that involves various traversal mechanism
• Realizes double polymorphic dispatch (multiple polymorphism)
Using Visitor in the Running Example

• The original problem statement does not require the use of this pattern
• Suppose we have a new requirement that asks us to implement undo and redo for each command (loadAlg, addDoc, removeDoc, query)
• If we hard code the undo and redo mechanism within each command type, we will have more difficulty to modify the undo / redo algorithm in the future
• Therefore, we want to implement undo / redo separately from our command objects
• Using Visitor pattern, we can accomplish these goals
• Describe how you would do it using slide 6