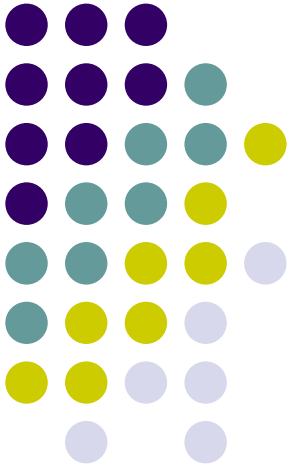
# Patrones de Diseño

## Patrón de comportamiento *Visitor*



# *Visitor*
## Propósito

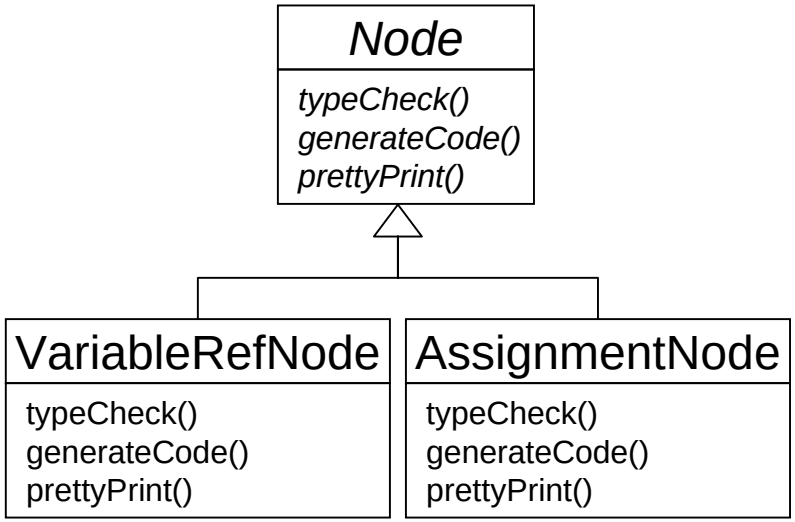

- Representa una operación a realizar sobre los elementos de una estructura de objetos

- Permite definir una nueva operación sin cambiar las clases de elementos sobre las que opera

# *Visitor*
## Motivación


- Ej: Un compilador representa los programas como árboles de sintaxis abstracta, sobre los que ejecuta operaciones

- Muchas operaciones necesitan diferenciar distintos tipos de nodo en el árbol (expresiones, variables, etc.)



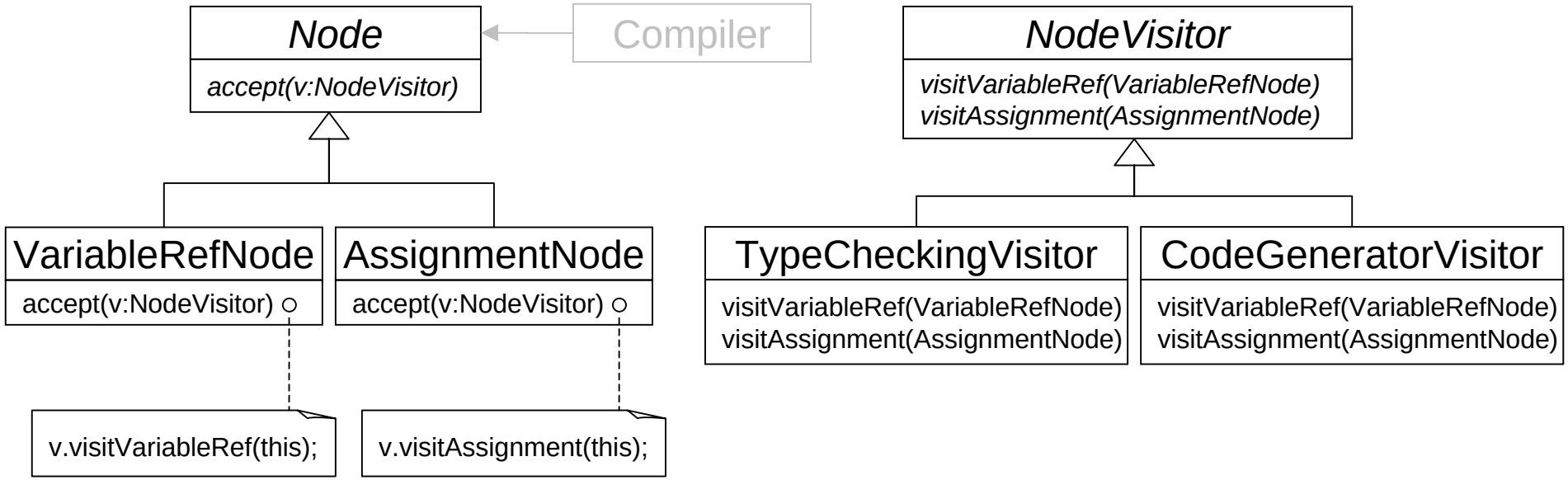

- Problemas:
  - difícil de comprender, mantener y cambiar
  - nuevas operaciones requerirán recompilar todas las clases

- Solución: independizar las clases de las operaciones que se ejecutan sobre ellas

# *Visitor*
## Motivación


Elementos sobre los que se opera

Operaciones

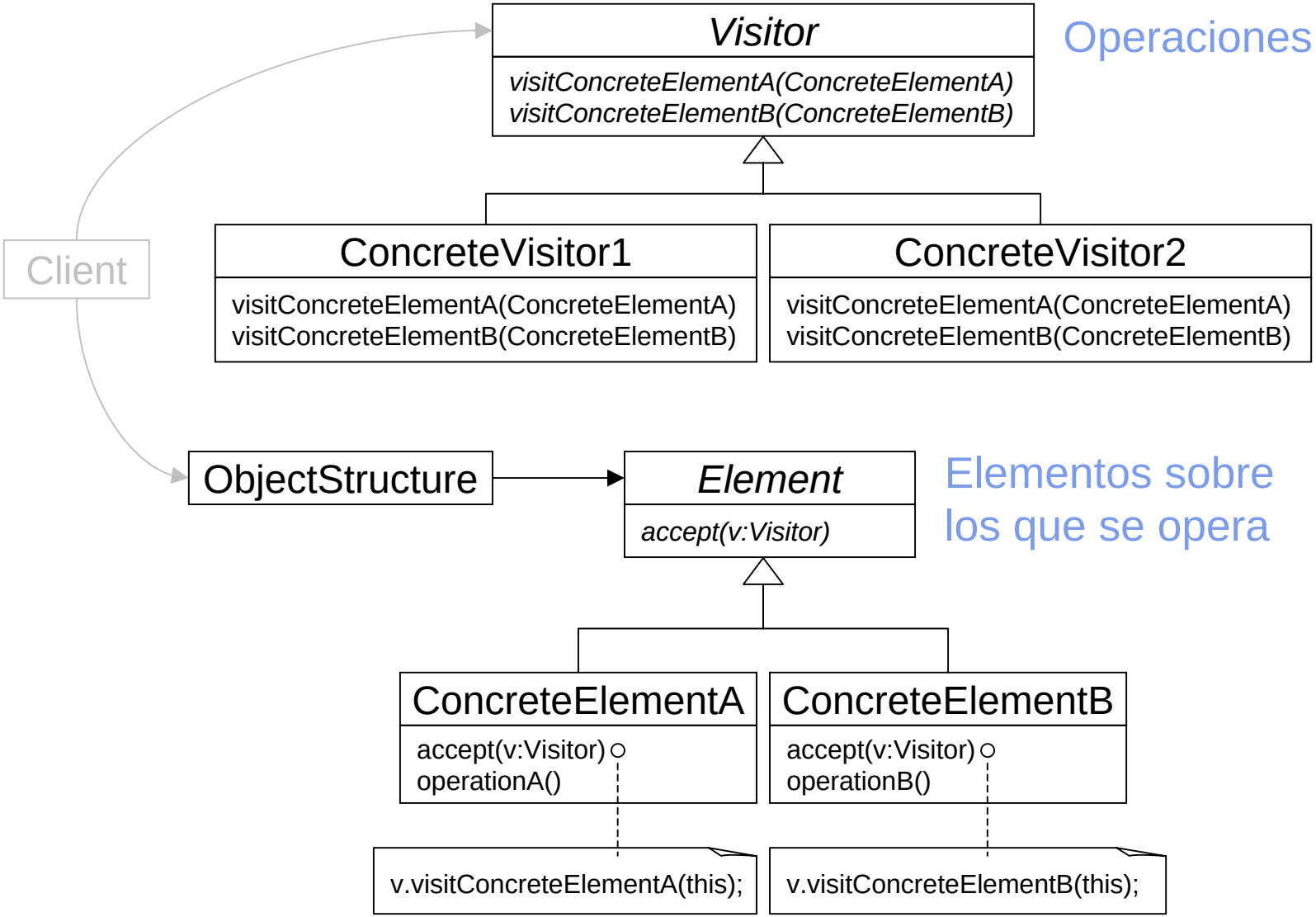


# *Visitor*
## Aplicabilidad


- Usa el patrón *Visitor* cuando:
  - Una estructura de objetos contiene muchas clases de objetos con interfaces distintas, y se quiere realizar sobre ellos operaciones que son distintas en cada clase concreta
  - Se quieren realizar muchas operaciones distintas sobre los objetos de una estructura, sin incluir dichas operaciones en las clases
  - Las clases que forman la estructura de objetos no cambian, pero las operaciones sobre ellas sí

# *Visitor*

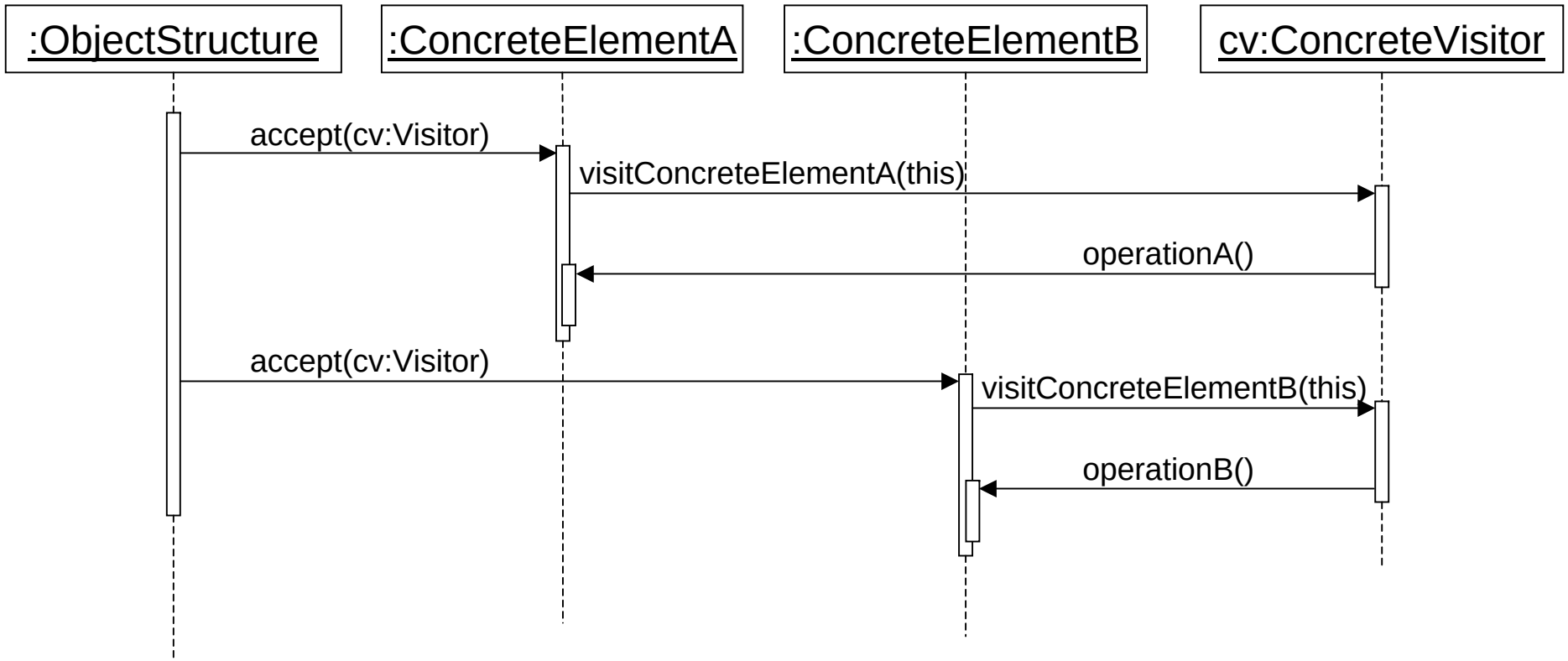
## Estructura



# *Visitor*
## Participantes


- **Visitor** (NodeVisitor): define una operación de visita para cada clase de elemento concreto en la estructura de objetos
- **ConcreteVisitor** (TypeCheckingVisitor):
  - Implementa la interfaz *Visitor*
  - Cada operación implementa un fragmento de la labor global del *visitor* concreto, pudiendo almacenar información local
- **Element** (Node)*:* define una operación *accept* con un *visitor* como argumento
- **ConcreteElement** (AssignmentNode)*:* implementa la operación *accept*
- **ObjectStructure** (Compiler):
  - Gestiona la estructura de objetos, y puede enumerar sus elementos
  - Puede ser un compuesto (patrón *composite*) o una colección de objetos
  - Puede ofrecer una interfaz que permita al *visitor* visitar a sus elementos

# *Visitor*
## Colaboraciones



# *Visitor*
## Consecuencias


- Facilita la definición de nuevas operaciones
- Agrupa operaciones relacionadas
- Añadir nuevas clases *ConcreteElement* es costoso
  - Utilizar el patrón *visitor* si la jerarquía de clases es estable
- Permite atravesar jerarquías de objetos que no están relacionados por un padre común
- El visitor puede acumular el estado de una operación al visitar la estructura de objetos, en vez de pasarlo como argumento o usar variables globales
- Rompe la encapsulación

# *Visitor*
## Implementación


- *Double dispatch*: técnica que permite añadir operaciones a las clases sin tener que modificarlas
  - La operación a ejecutar depende de la clase de petición (*accept*) y del tipo de los dos receptores (*Visitor* y *Element*)

- ¿Quién es responsable de recorrer la estructura de objetos?
  - La estructura de objetos (ej. composite)
  - Un iterador interno o externo
  - El *visitor*: duplica el código de recorrido en cada objeto de tipo compuesto. Sólo se utiliza para implementar recorridos complejos que dependen de los resultados de las operaciones.