

---

# Patrón Visitor

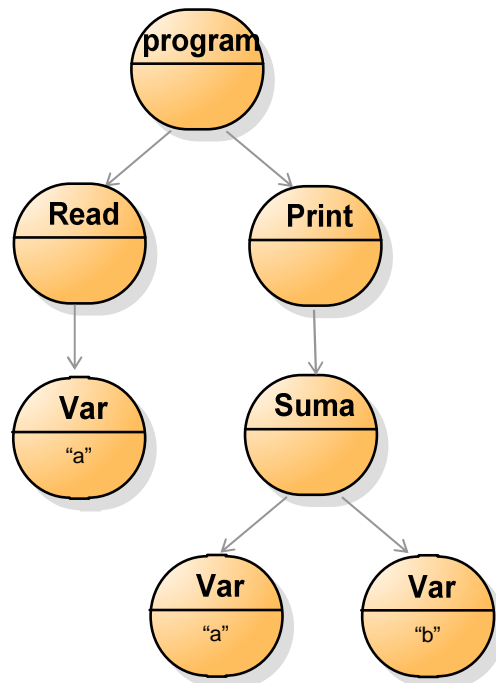
---

**Diseño de Software (v1.8)**  
**Raúl Izquierdo Castanedo**

# Modelo de ejemplo

## Ejemplo

```
read a;  
print a + b;
```



## Nodos del árbol

```
interface Nodo { }  
  
class Programa implements Nodo {  
    List<Sentencia> sentencias;  
}  
  
interface Sentencia extends Nodo { }  
  
class Read implements Sentencia {  
    Variable var;  
}  
class Print implements Sentencia {  
    Expression expr;  
}  
  
interface Expression extends Nodo { }  
  
class Suma implements Expression {  
    Expression left, right;  
}  
class Variable implements Expression {  
    String name;  
}
```

---

# Soporte de Recorridos del Modelo

## Se desea recorrer los objetos con distintos objetivos:

- Imprimir el programa
- Análisis Semántico
- Compilar
- Documentar (javaDoc)
- Ejecutar
- Y en un futuro...

## Formas de implementar el código de recorrido:

1. Repartir el código del recorrido entre los nodos
  - ❑ Mejor cuando los recorridos son más estables que los nodos
    - Patrón *Interpreter*
2. Centralizarlo en un objeto
  - ❑ Mejor cuando los nodos son más estables que los recorridos
    - Patrón *Visitor*

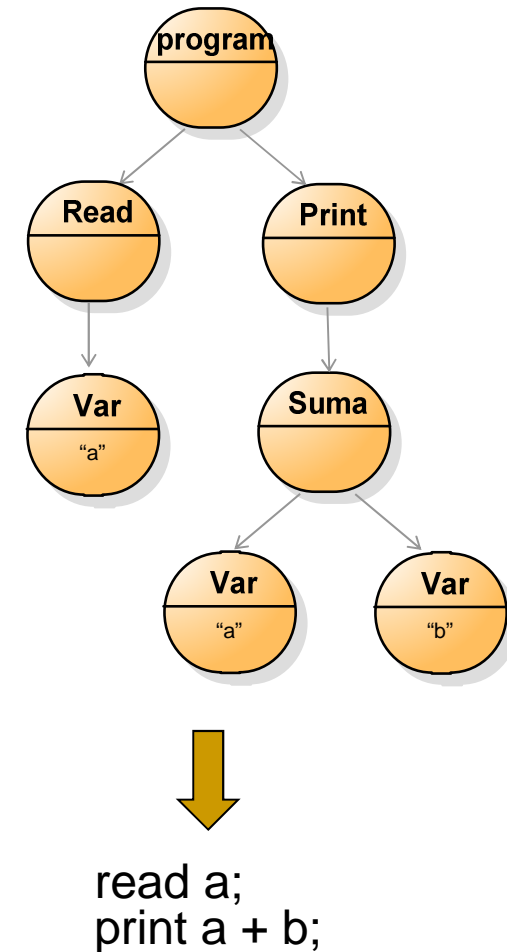
## Nos centraremos en el segundo caso

- A lo largo del desarrollo
  - ❑ Los objetos nodos serán más estables
  - ❑ Se añadirán y quitarán distintos recorridos
    - Queremos poder hacerlo sin modificar los nodos!!!

# Recorrido Recursivo

```
public static void main(String[] args) {  
    Programa prog = new Programa ... // Construir aquí el árbol  
  
    RecorridoRecursivo recorrido = new RecorridoRecursivo();  
    recorrido.visit(prog);  
}
```

```
public class RecorridoRecursivo {  
    public void visit(Nodo nodo) {  
        if (nodo instanceof Programa) {  
            for (Sentencia sent : ((Programa) nodo).sentencias)  
                visit(sent);  
        } else if (nodo instanceof Print) {  
            System.out.println("print ");  
            visit( ((Print)nodo).expr );  
            System.out.println(";");  
        } else if (nodo instanceof Read) {  
            System.out.println("read ");  
            visit( ((Read)nodo).var );  
            System.out.println(";");  
        } else if (nodo instanceof Suma) {  
            visit( ((Suma)nodo).left );  
            System.out.println("+");  
            visit( ((Suma)nodo).right );  
        } else if (nodo instanceof Variable) {  
            System.out.println( ((Variable)nodo).name );  
        }  
    }  
}  
  
(*)
```



# Recorrido con un método por nodo

## Versión Ideal

```
public class PrintPrograma          // Versión ideal
{
    public void visit(Programa programa) {
        for (Sentencia sent : programa.sentencias)
            visit(sent);
    }

    public void visit(Print print) {
        System.out.println("print ");
        visit(print.expr);
        System.out.println(";");
    }

    public void visit(Read read) {
        System.out.println("read ");
        visit(read.var);
        System.out.println(";");
    }

    public void visit(Suma suma) {
        visit(suma.left);
        System.out.println(" + ");
        visit(suma.right);
    }

    public void visit(Variable var) {
        System.out.println(var.name);
    }
}
```

¡No compila!

# Visitor con un método por nodo

```
interface Figura
{
}

class Circulo implements Figura
{
}
```

```
class Prueba
{
    void imprime(Figura f) {
        System.out.println("Figura");
    }

    void imprime(Circulo c) {
        System.out.println("Circulo");
    }

    public static void main(String[] args){
        Figura circulo = new Circulo();
        imprime(circulo);    // ¿Qué sale?
    }
}
```

- **Hay lenguajes que disponen de esta característica**
  - Multiple dispatch

# Visitor con un método por nodo

```
public class PrintPrograma    // versión ideal
{
    public void visit(Programa prog) {
        for (Sentencia sent : prog.sentencias)
            visit(sent);
    }

    public void visit(Print print) {
        System.out.println("print ");
        visit(print.expr);
        System.out.println(";");
    }

    public void visit(Read read) {
        System.out.println("read ");
        visit(read.var);
        System.out.println(";");
    }

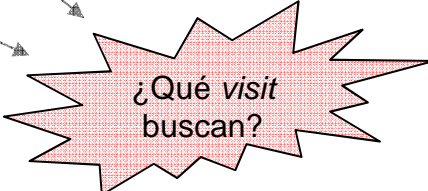
    public void visit(Suma suma) {
        visit(suma.left);
        System.out.println(" + ");
        visit(suma.right);
    }

    public void visit(Variable var) {
        System.out.println(var.name);
    }
}
( *)
```

```
class Programa implements Nodo {
    List<Sentencia> sentencias;
}
```

```
class Print implements Sentencia {
    Expression expr;
}
```

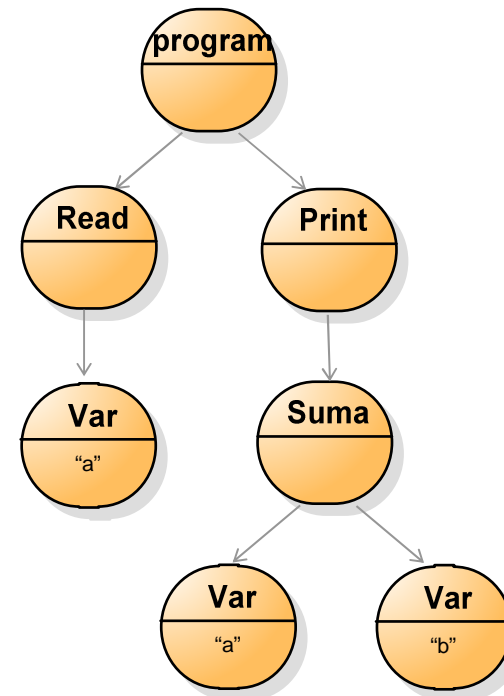
```
class Suma implements Expression {
    Expression left, right;
}
```



¿Qué visit  
buscan?

# Objetivo

```
void visit(Print print) {  
    System.out.println("print ");  
    visit(print.expr);  
    System.out.println(";");  
}  
  
void visit(Suma suma) {  
    visit(suma.left);  
    System.out.println(" + ");  
    visit(suma.right);  
}  
  
void visit(Variable var) {  
    System.out.println(var.name);  
}
```



- ¿Volvemos a los *if/else* con *instanceof*?

Sí... por favor...



# Solución: Patrón Visitor

```
public static void main(String[] args) {
    Programa prog = new Programa ... // Construir aquí el árbol

    PrintVisitor visitor = new PrintVisitor();
    prog.accept(visitor);
}
```

Interfaz con un método para cada nodo

```
public interface Visitor {
    void visitProg(Programa p);
    void visitPrint(Print p);
    void visitRead(Read r);
    void visitSuma(Suma s);
    void visitVariable(Variable v);
}
```

Son los nodos los que eligen el método adecuado

```
public interface Nodo {
    void accept(Visitor v);
}
```

Redefiniendo el método `accept` se elige el `visit` correspondiente al nodo

```
public class Print implements Sentencia {
    ...
    public void accept(Visitor v) {
        v.visitPrint(this);
    }
}

public class Read implements Sentencia {
    ...
    public void accept(Visitor v) {
        v.visitRead(this);
    }
}
```

public class **PrintVisitor** implements **Visitor** {

```
    public void visitProg(Programa prog) {
        for (Sentencia sent : prog.sentencias)
            sent.accept(this);
    }
```

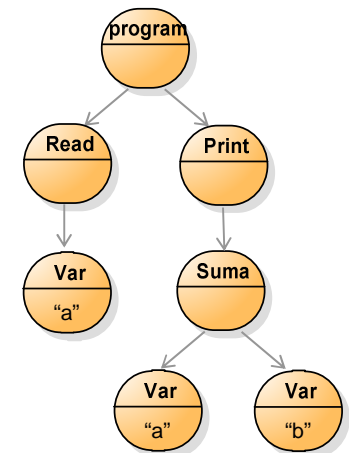
Desde un método *visit* siempre se llama a *accept* (nunca a otro *visit*)

```
    public void visitPrint(Print print) {
        System.out.print("print ");
        print.expr.accept(this);
        System.out.println(";");
    }
```

```
    public void visitRead(Read read) {
        System.out.print("read ");
        read.var.accept(this);
        System.out.println(";");
    }
```

```
    public void visitSuma(Suma suma) {
        suma.left.accept(this);
        System.out.print(" + ");
        suma.right.accept(this);
    }
```

```
    public void visitVariable(Variable var) {
        System.out.print(var.name);
    }
}
```



# Opcional: Unificar nombres (sobrecarga)

No se necesita que los nombres sean distintos

```
public interface Visitor {  
    void visitProg(Programa p);  
    void visitPrint(Print p);  
    void visitRead(Read r);  
    void visitSuma(Suma s);  
    void visitVariable(Variable v);  
}
```

El Nodo no cambia

```
public interface Nodo {  
    void accept(Visitor v);  
}
```

Pero ahora todos los métodos `accept` son iguales!!

```
public class Print implements Sentencia {  
    ...
```

```
    public void accept(Visitor v) {  
        v.visitPrint(this);  
    }  
}
```

```
public class Read implements Sentencia {  
    ...
```

```
    public void accept(Visitor v) {  
        v.visitRead(this);  
    }  
}
```

Se puede copiar y pegar  
en todos los nodos

```
public class PrintVisitor implements Visitor {
```

```
    public void visitProg(Programa prog) {  
        for (Sentencia sent : prog.sentencias)  
            sent.accept(this);  
    }
```

```
    public void visitPrint(Print print) {  
        System.out.print("print ");  
        print.expr.accept(this);  
        System.out.println(";");  
    }
```

```
    public void visitRead(Read read) {  
        System.out.print("read ");  
        read.var.accept(this);  
        System.out.println(";");  
    }
```

```
    public void visitSuma(Suma suma) {  
        suma.left.accept(this);  
        System.out.print(" + ");  
        suma.right.accept(this);  
    }
```

```
    public void visitVariable(Variable var) {  
        System.out.print(var.name);  
    }  
}
```

# Generalizando el Patrón Visitor

## El nodo debe poder ser recorrido para cualquier tarea

- Alguna podría requerir parámetros y/o valores de retorno

### Generalizando los nodos...

```
public interface Nodo {
    Object accept(Visitor v, Object param);
}

public class Print implements Sentencia {
    ...
    public Object accept(Visitor v, Object param) {
        return v.visit(this, param);
    }
}

public class Read implements Sentencia {
    ...
    public Object accept(Visitor v, Object param) {
        return v.visit(this, param);
    }
}
```

### Generalizando el Visitor...

```
public interface Visitor {
    Object visit(Programa p, Object param);
    Object visit(Print p, Object param);
    Object visit(Read r, Object param);
    Object visit(Suma s, Object param);
    Object visit(Variable v, Object param);
}
```

## Implementando el nuevo Visitor...

- Ejemplo de cómo implementarlo cuando no se necesiten el nuevo parámetro y el valor de retorno

```
public class PrintVisitor implements Visitor {
    public Object visit(Programa prog, Object param) {
        for (Sentencia sent : prog.sentencias)
            sent.accept(this, null);
        return null;
    }

    public Object visit(Print print, Object param) {
        System.out.print("print ");
        print.expr.accept(this, null);
        System.out.println(";");
        return null;
    }

    public Object visit(Read read, Object param) {
        System.out.print("read ");
        read.var.accept(this, null);
        System.out.println(";");
        return null;
    }

    public Object visit(Suma suma, Object param) {
        suma.left.accept(this, null);
        System.out.print(" + ");
        suma.right.accept(this, null);
        return null;
    }

    public Object visit(Variable var, Object param){
        System.out.print(var.name);
        return null;
    }
}
```

# Resumen

## Pasos para implementar el patrón Visitor

1) Hacer un interfaz **Visitor** con un método *visit* por cada nodo

```
public interface Visitor {  
    public Object visit(Programa p, Object param);  
    public Object visit(Print p, Object param);  
    ...  
}
```

2) Hacer un interfaz AST con un método **accept**

```
public interface AST {  
    Object accept(Visitor v, Object param);  
}
```

3) Hacer que todos los nodos implementen el interfaz AST y en el método *accept* que invoque al método **visit** adecuado

```
class Print implements Sentencia {  
    ...  
    Object accept(Visitor v, Object param) {  
        return v.visit(this, param);  
    }  
}  
  
class Read implements Sentencia {  
    ...  
    Object accept(Visitor v, Object param) {  
        return v.visit(this, param);  
    }  
}
```

4) Para implementar un nuevo recorrido hay que dar implementación a todos los métodos del interfaz *Visitor*

```
public class PrintVisitor implements Visitor {  
    ...  
}
```