Compiler Construction

Visitors

Goals

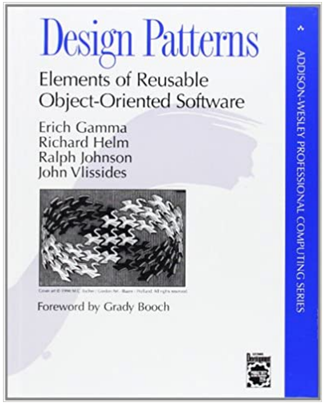
Primary Goal

How to separate an algorithm from the structure on which it operates?

In this course

How to have an external processing of the AST?

23 classic software design patterns


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides [1994]

A.k.a. The Gang of Four

Visitors: description

Define new operations without changing the classes of the elements on which we operate.

In essence, the visitor patterns allows adding new virtual functions to a family of classes

Solution to multiple dispatch (multimethods) when the language does not provide it.

Main Idea

Augment all (AST) classes with an **accept** method

Group all processing into a single **visitor** class, defining a visit method for each AST class which calls accept on the visitor.

accept performs the dynamic dispatch and calls the correct callback!

Example (1/3)

```cpp
class PrettyPrinter
{
public:
  void visitNum(Num& e) {
    ostr_ << e.val();
  }
  void visitBin(Bin& e) {
    ostr_ << '(';
    e.lhs()->accept(*this);
    // ...
  }
private:
  std::ostream& ostr_;
  unsigned tab_;
};
```

Example (2/3)

```
class Exp {
public:
  virtual void accept(Visitor& v) const = 0;
};
```

Example (3/5)

```
class Num : public Exp {
public:
  Num(int val)
    : Exp(), val_(val)
  {}

  void accept(Visitor& v) const override {
    v.visitNum(*this);
  }

private:
  int val_;
};
```

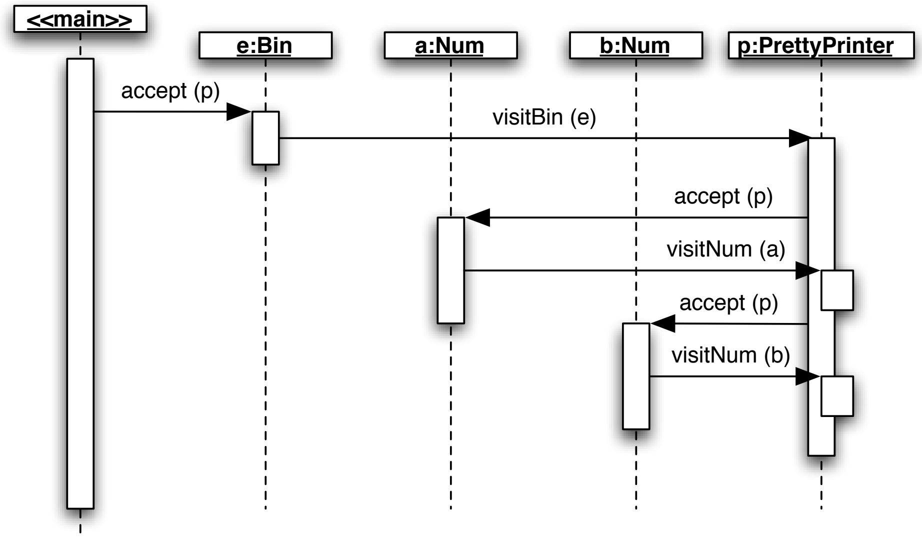
Example (4/5)

```
class Bin : public Exp
{
public:
  // Constructors, Destructors, and getters

  void accept(Visitor& v) const override {
    v.visitBin(*this);
  }

private:
  char oper_;
  Exp* lhs_; Exp* rhs_;
};
```

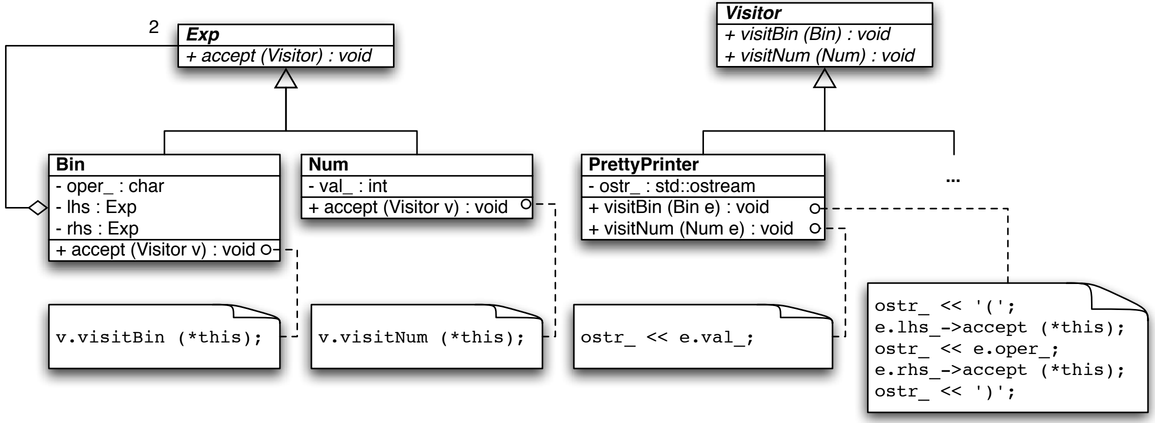
Example (5/5)

```cpp
int main()
{
  Bin* bin = new Bin(
    '+',
    new Num(42),
    new Num(51)
  );
  Exp* exp = bin;

  auto printer = PrettyPrinter{std::cout};
  exp.accept(printer);
  delete bin;
}
```

Sequence Diagram



A class diagram: Visitor and Composite Patterns



```
v.visitBin (*this);
```

```
v.visitNum (*this);
```

```
ostr_ << e.val_;
```

```
ostr_ << '(';
e.lhs_->accept (*this);
ostr_ << e.oper_;
e.rhs_->accept (*this);
ostr_ << ')';
```

Using operator<<

```
std::ostream& operator<<(std::ostream& o, const Exp& e)
{
  auto printer = PrettyPrinter{o};
  e.accept(printer);
  return o;
}

int main() {
  Bin* bin = new Bin(/*..*/);
  std::cout << *bin;
  delete bin;
}
```

Summary

visitor

composite

accept

visit

design patterns