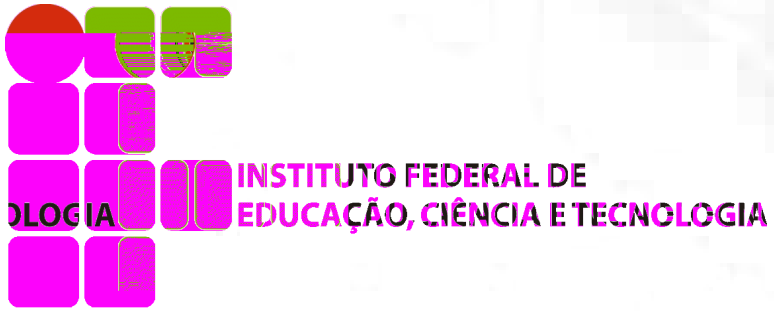
Interpreter

Sandro Santos Andrade

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas



Interpreter

-
  -
-
  -
  -
  -

strings

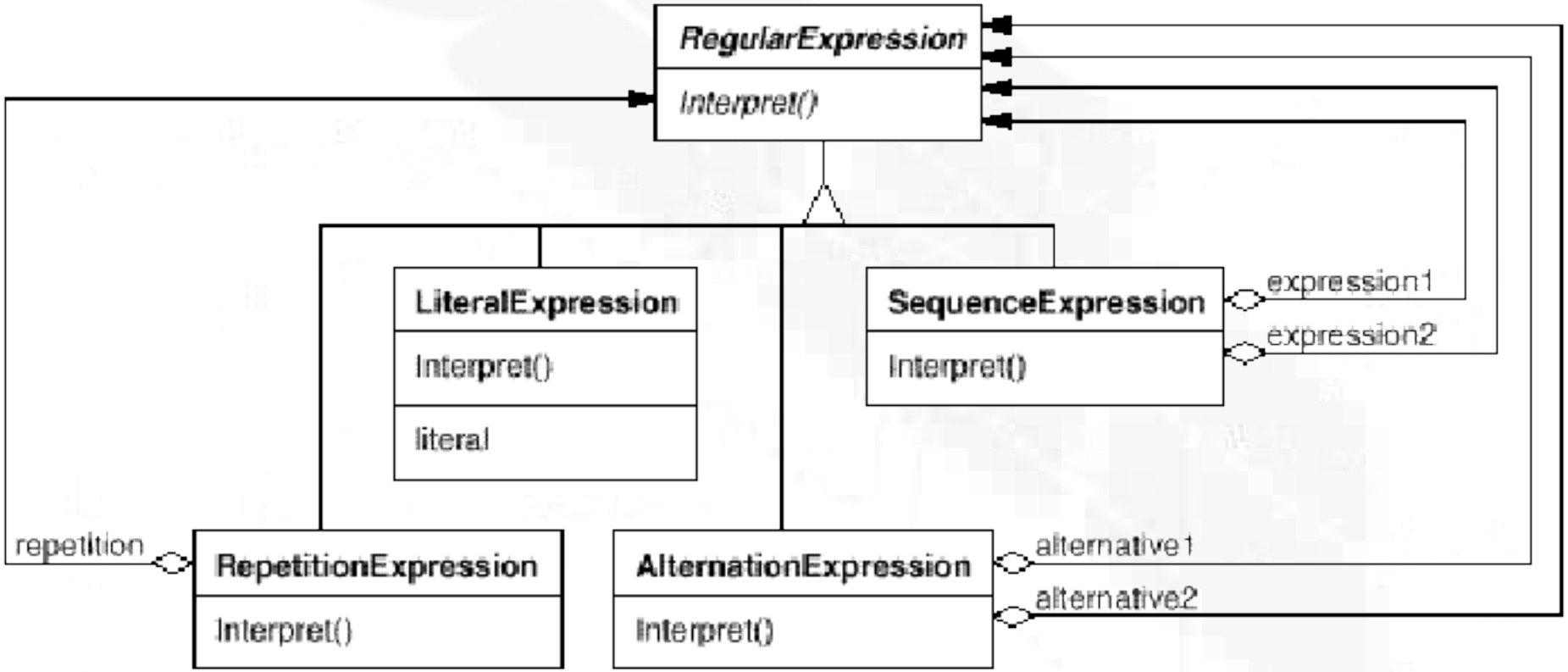
Interpreter

-
  -

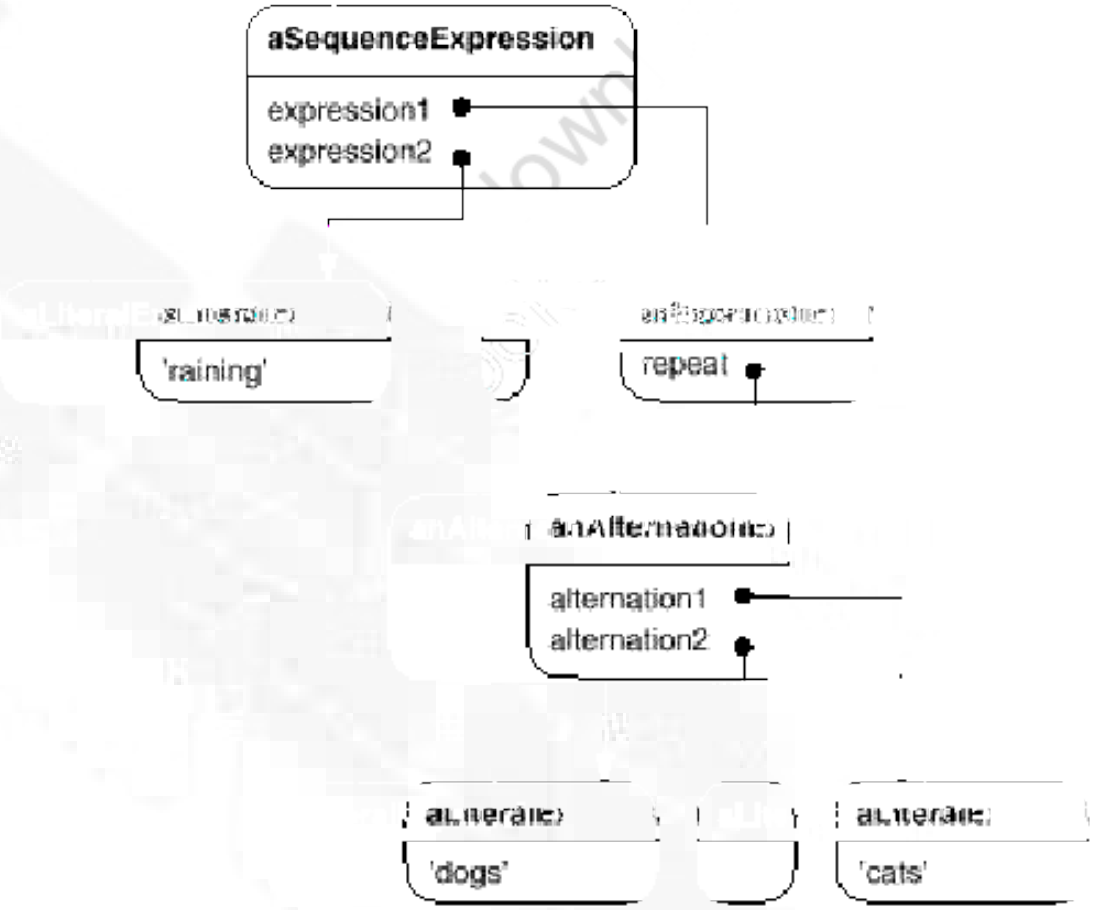
```
expression ::= literal | alternation | sequence | repetition |
               '(' expression ')'
alternation ::= expression  '|' expression
sequence ::= expression '&' expression
repetition ::= expression '*'
literal ::= 'a' | 'b' | 'c' | 'd' | 'e'
```

    -
    -
    - RegularExpression
      LiteralExpression
      AlternationExpression SequenceExpression
      RepetitionExpression

Interpreter



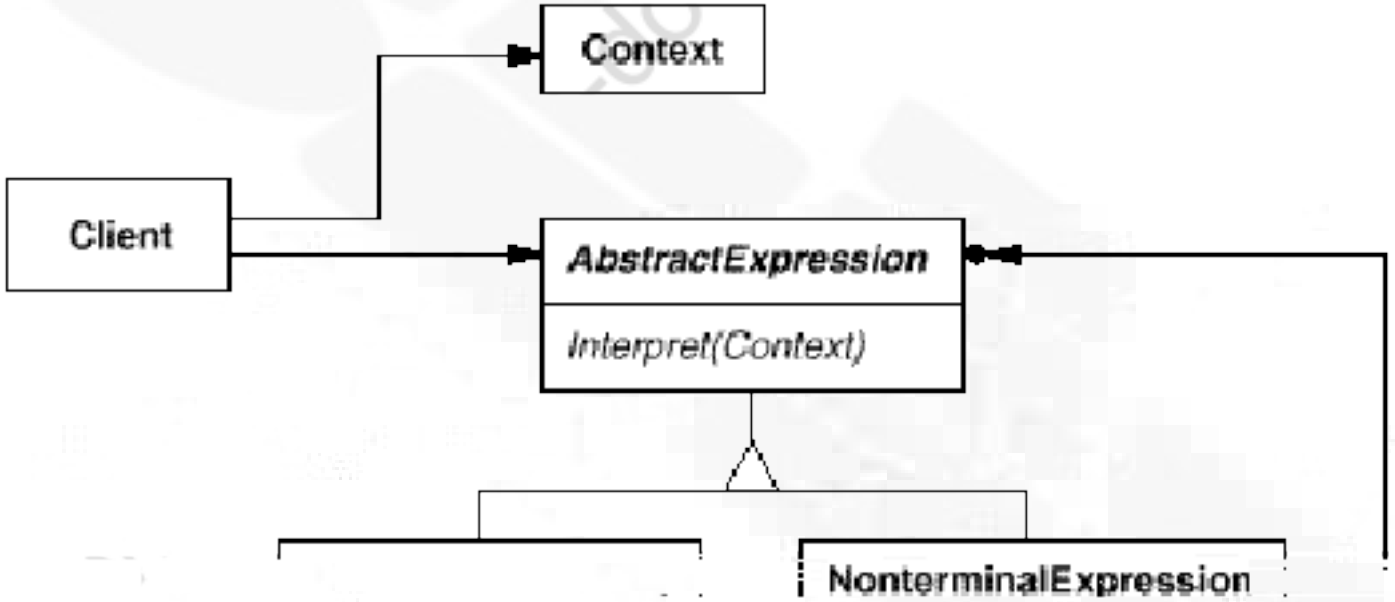
Interpreter



Interpreter

-
  -
    parsers
  -

Interpreter



Interpreter

-
  - AbstractExpression
    - interpret()
  - TerminalExpression
    - interpret()
    -

Interpreter

-
  - NonTerminalExpression
    -
    - AbstractExpression
    - interpret()

interpret()

Interpreter

-
  - Context
    -
  - Client
    -
      - interpret()

Interpreter

Interpreter

-
  -
    -
    - parsers
  -
    -
    - type-c ec*!*ing
    - ”isitor

Interpreter

-
  -
    - Interpreter
      parsing
    - parser
  - interpret()
    - interpret()
      ″isitor
  - #ly $eig t
    -

Interpreter

```
BooleanExp ::= VariableExp | Constant | OrExp | AndExp | NotExp |
               '(' BooleanExp ')'
AndExp ::= BooleanExp  'and' BooleanExp
OrExp ::= BooleanExp  'or' BooleanExp
NotExp ::= 'not' BooleanExp
Constant ::= 'true' |  'false'
VariableExp ::= 'A' | 'B' | ... | 'X' | 'Y' | 'Z'
```

Interpreter

```
class BooleanExp {
public:
    BooleanExp();
    virtual ~BooleanExp();
    virtual bool Evaluate(Context&) = 0;
    virtual BooleanExp* Replace(const char*, BooleanExp&) = 0;
    virtual BooleanExp* Copy() const = 0;
};
```

Interpreter

```
class Context {
public:
    bool Lookup(const char*) const;
    void Assign(VariableExp*, bool);
};
```

```
class VariableExp : public BooleanExp {
public:
    VariableExp(const char*);
    virtual ~VariableExp();

    virtual bool Evaluate(Context&);
    virtual BooleanExp* Replace(const char*, BooleanExp&);
    virtual BooleanExp* Copy() const;
private:
    char* _name;
};
```

Interpreter

```
VariableExp::VariableExp (const char* name) {
    _name = strdup(name);
}
```

```
bool VariableExp::Evaluate (Context& aContext) {
    return aContext.Lookup(_name);
}
```

```
BooleanExp* VariableExp::Copy () const {
    return new VariableExp(_name);
}
```

Interpreter

```
BooleanExp* VariableExp::Replace (
    const char* name, BooleanExp& exp
) {
    if (strcmp(name, _name) == 0) {
        return exp.Copy();
    } else {
        return new VariableExp(_name);
    }
}
```

Interpreter

```
class AndExp : public BooleanExp {
public:
    AndExp(BooleanExp*, BooleanExp*);
    virtual ~ AndExp();

    virtual bool Evaluate(Context&);
    virtual BooleanExp* Replace(const char*, BooleanExp&);
    virtual BooleanExp* Copy() const;
private:
    BooleanExp* _operand1;
    BooleanExp* _operand2;
};

AndExp::AndExp (BooleanExp* op1, BooleanExp* op2) {
    _operand1 = op1;
    _operand2 = op2;
}
```

Interpreter

```
bool AndExp::Evaluate (Context& aContext) {
    return
        _operand1->Evaluate(aContext) &&
        _operand2->Evaluate(aContext);
}
```

```
BooleanExp* AndExp::Copy () const {
    return
        new AndExp(_operand1->Copy(), _operand2->Copy());
}

BooleanExp* AndExp::Replace (const char* name, BooleanExp& exp) {
    return
        new AndExp(
            _operand1->Replace(name, exp),
            _operand2->Replace(name, exp)
        );
}
```

Interpreter

```
BooleanExp* expression;
Context context;
VariableExp* x = new VariableExp("X");
VariableExp* y = new VariableExp("Y");

expression = new OrExp(
    new AndExp(new Constant(true), x),
    new AndExp(y, new NotExp(x))
);

context.Assign(x, false);
context.Assign(y, true);

bool result = expression->Evaluate(context);
```

Interpreter

```
VariableExp* z = new VariableExp("Z");
NotExp not_z(z);

BooleanExp* replacement = expression->Replace("Y", not_z);

context.Assign(z, true);

result = replacement->Evaluate(context);
```

Interpreter

Interpreter

-
  - Composite
  - #ly $eig t
  - Iterator
  - ″isitor

Interpreter

Sandro Santos Andrade

Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Departamento de Tecnologia Eletro-Eletrônica
Graduação Tecnológica em Análise e Desenvolvimento de Sistemas