

AST

Лекция №7

Пример MiniJavaGo

Примеры к лекции:

1. [Пример MiniJavaGo](#) — Go, стековая VM
2. [Пример PsTiger](#) — C#, интерпретация без VM и стековая VM
3. [Пример PsTigerCpp](#) — C++, компилятор

Цель

Абстрагировать разбор языка от дальнейших действий:

1. Генерация байт-кода VM
2. Интерпретация без VM
3. Работа LSP-сервера

Пример: интерпретатор выполняет программу

```
parser := parse.NewParser(source)
program, errs := parser.Compile()
if len(errs) > 0 {
    for _, err := range errs {
        env.PrintError(err)
    }
    return fmt.Errorf("failed to compile %q", sourcePath)
}

return vm.RunProgram(env, program)
```

Пример: LSP-сервер собирает диагностику

// Выполняем синтаксический разбор.

```
parser := parse.NewParser([]byte(text))
```

```
_, errs := parser.Compile()
```

// Преобразуем ошибки в формат Language Server Protocol.

```
collector := newDiagnosticsCollector(impl.serverName)
```

```
for _, e := range errs {
```

```
    collector.addError(e)
```

```
}
```

// Отправляем диагностику редактору.

```
context.Notify(protocol.ServerTextDocumentPublishDiagnostics, ...)
```

Пример: LSP-сервер собирает диагностику

```
// Выполняем синтаксический разбор.  
parser := parse.NewParser([]byte(text))  
_, errs := parser.Compile()
```

Продукция парсера:
1. Инструкции VM
2. Ошибки разбора



```
// Преобразуем ошибки в формат Language Server Protocol.  
collector := newDiagnosticsCollector(impl.serverName)  
for _, e := range errs {  
    collector.addError(e)  
}
```

```
// Отправляем диагностику редактору.  
context.Notify(protocol.ServerTextDocumentPublishDiagnostics, ...)
```

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	???

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	Дерево: • Parse Tree • AST — Abstract Syntax Tree

Parse Tree

Выражение

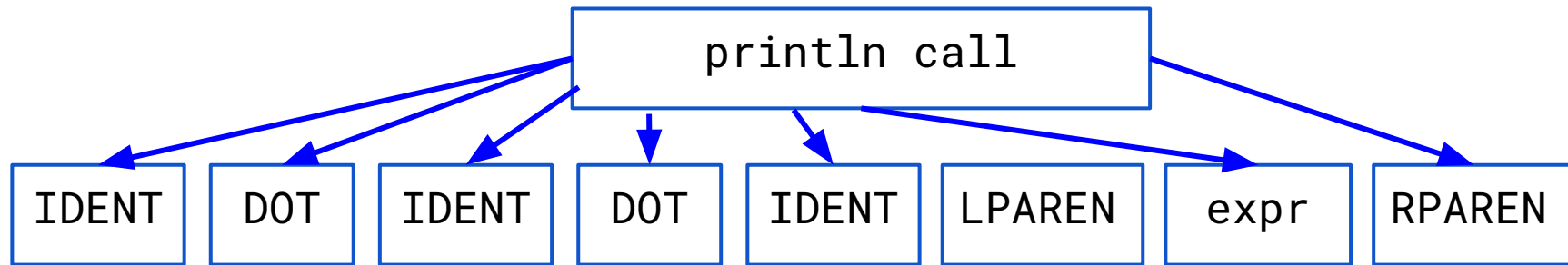
```
System.out.println(10 * (5 - 3))
```

Дерево разбора — Parse Tree

1. N-арное дерево
2. Узел — символ грамматики (терминал или нетерминал)
3. Узел хранит
 - идентификатор правила
 - список символов

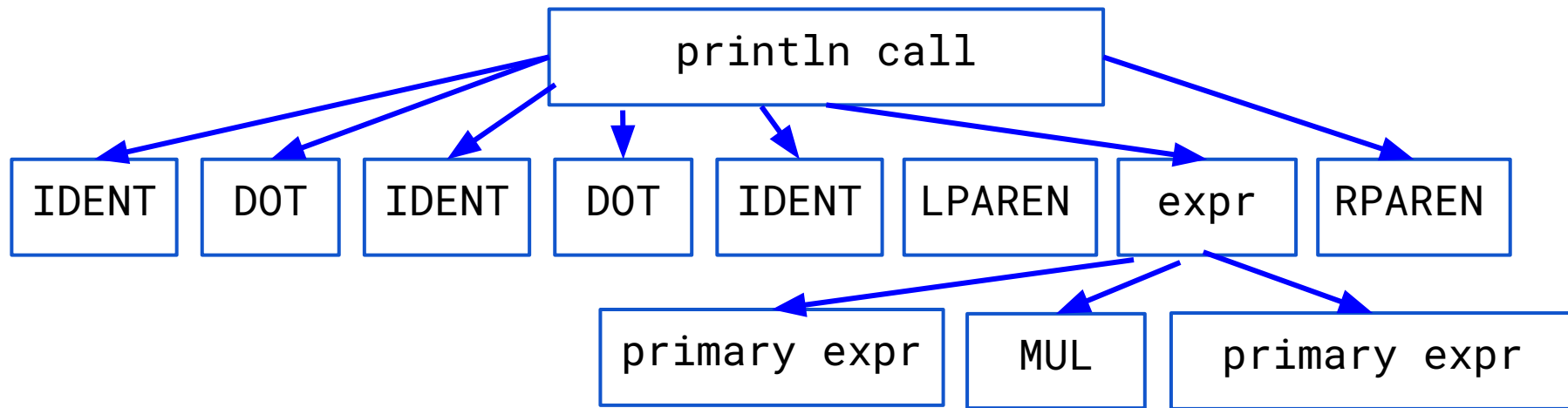
```
System.out.println(10 * (5 - 3))
```

Дерево разбора (Parse Tree)



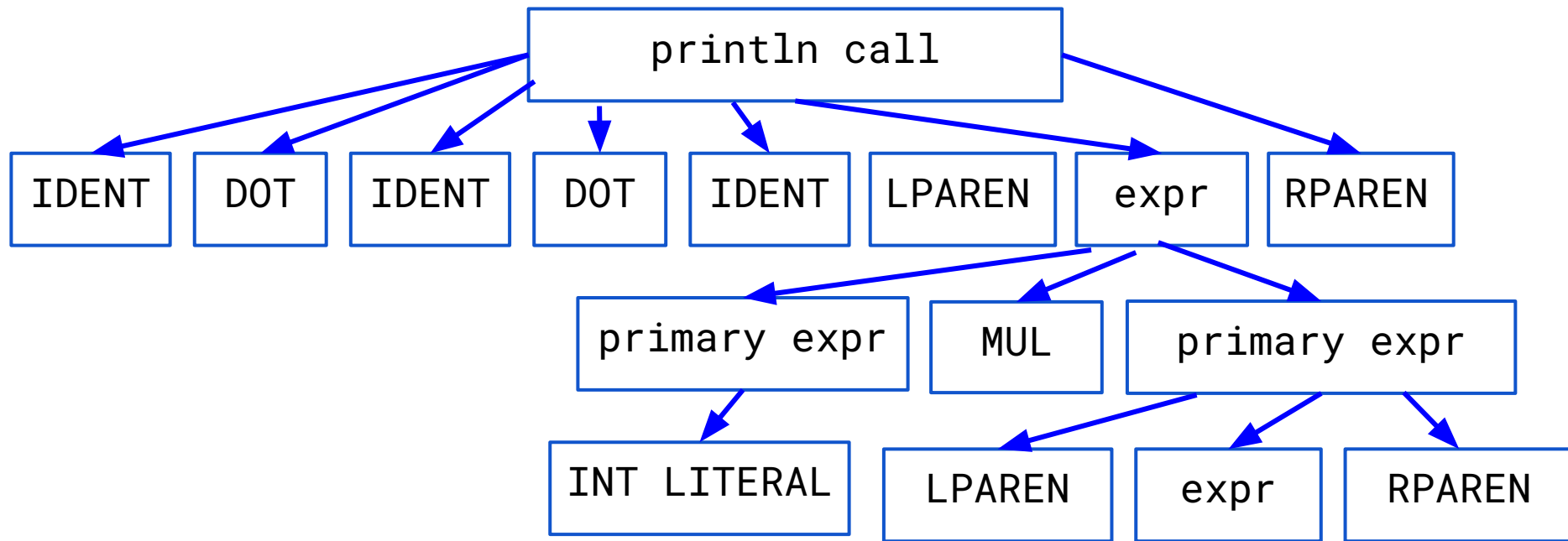
`System.out.println(10 * (5 - 3))`

Дерево разбора (Parse Tree)



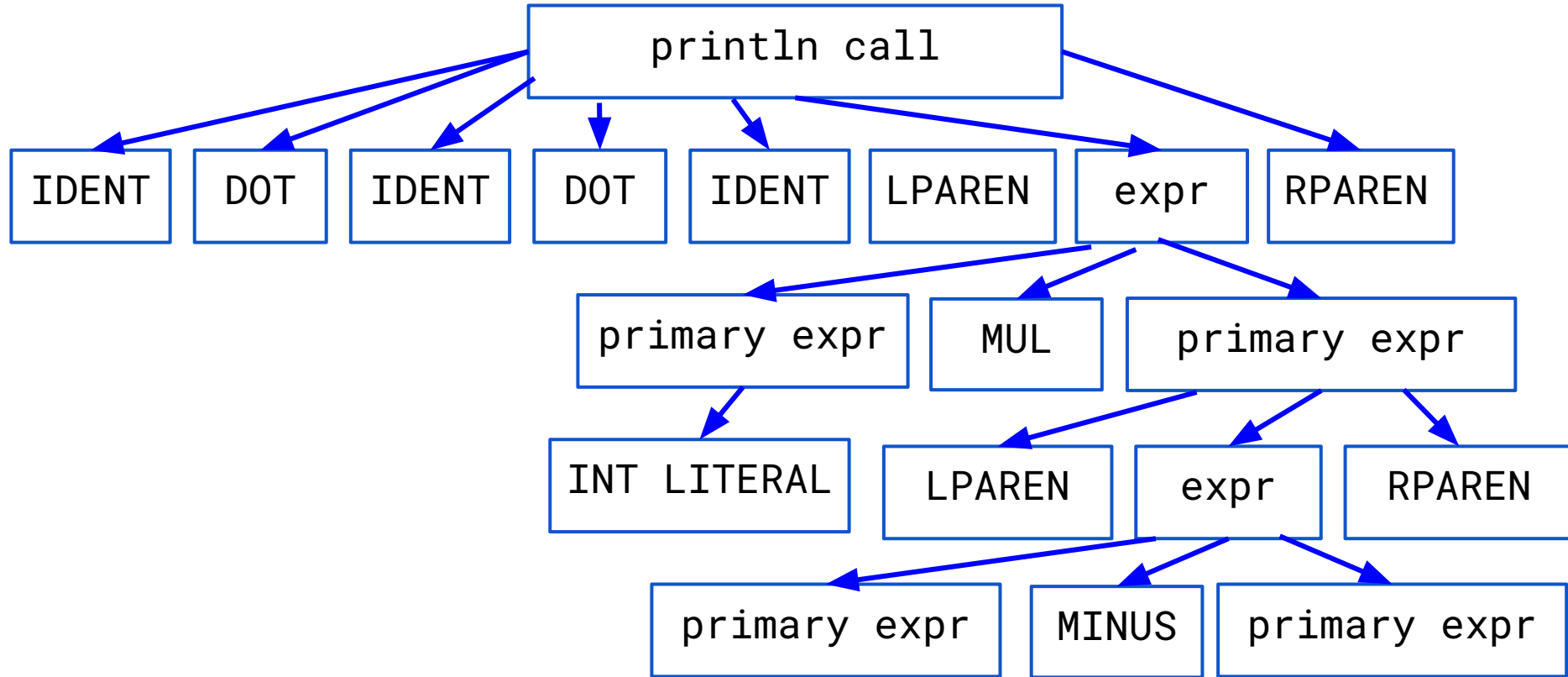
`System.out.println(10 * (5 - 3))`

Дерево разбора (Parse Tree)



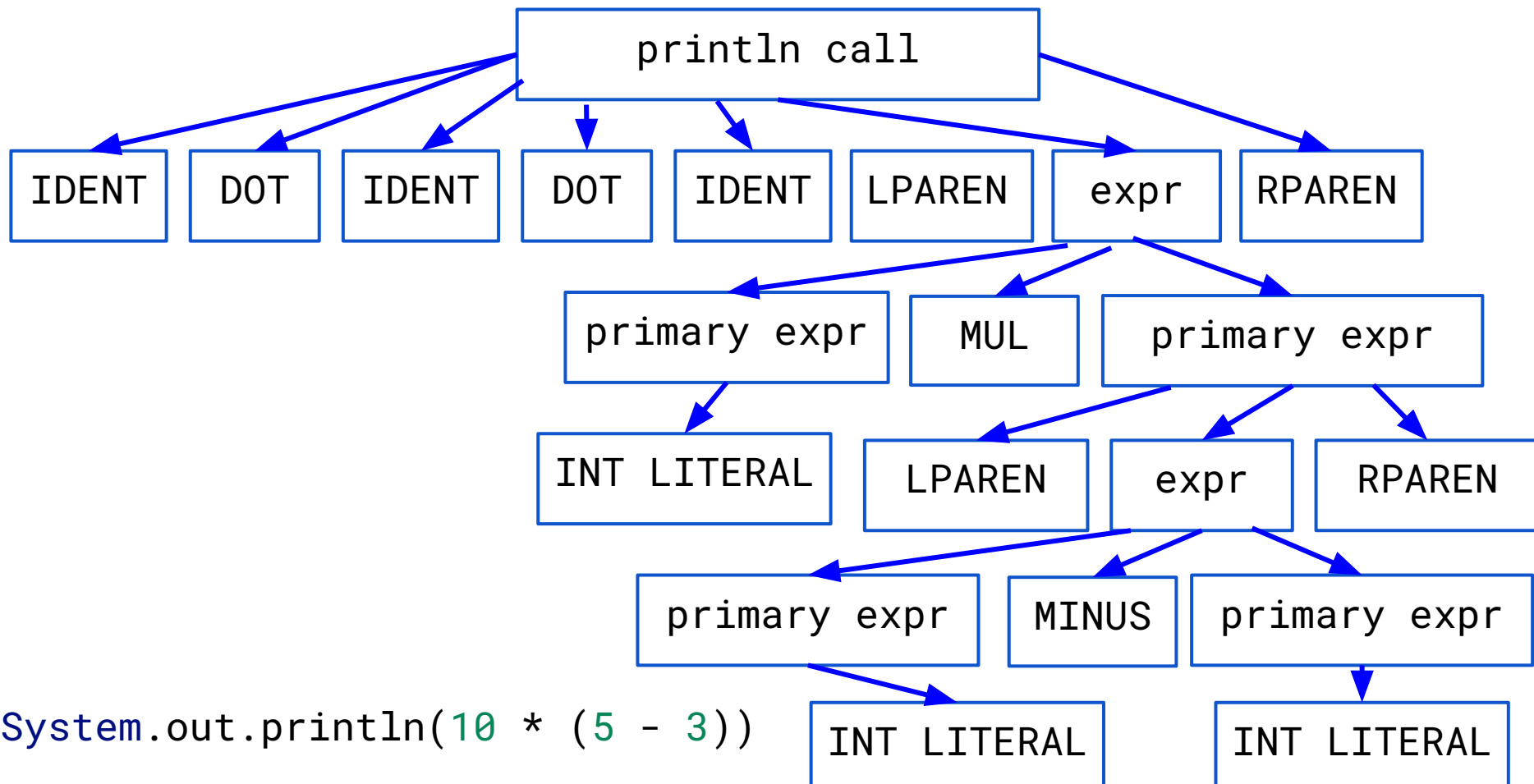
`System.out.println(10 * (5 - 3))`

Дерево разбора (Parse Tree)



`System.out.println(10 * (5 - 3))`

Дерево разбора (Parse Tree)



Дерево разбора — Parse Tree

1. N-арное дерево
2. Узел — символ грамматики (терминал или нетерминал)
3. Узел хранит
 - идентификатор правила
 - список символов

Проблемы:

- Избыточная информация о лексемах и синтаксисе
- Разные варианты одного правила

```
System.out.println(10 * (5 - 3))
```

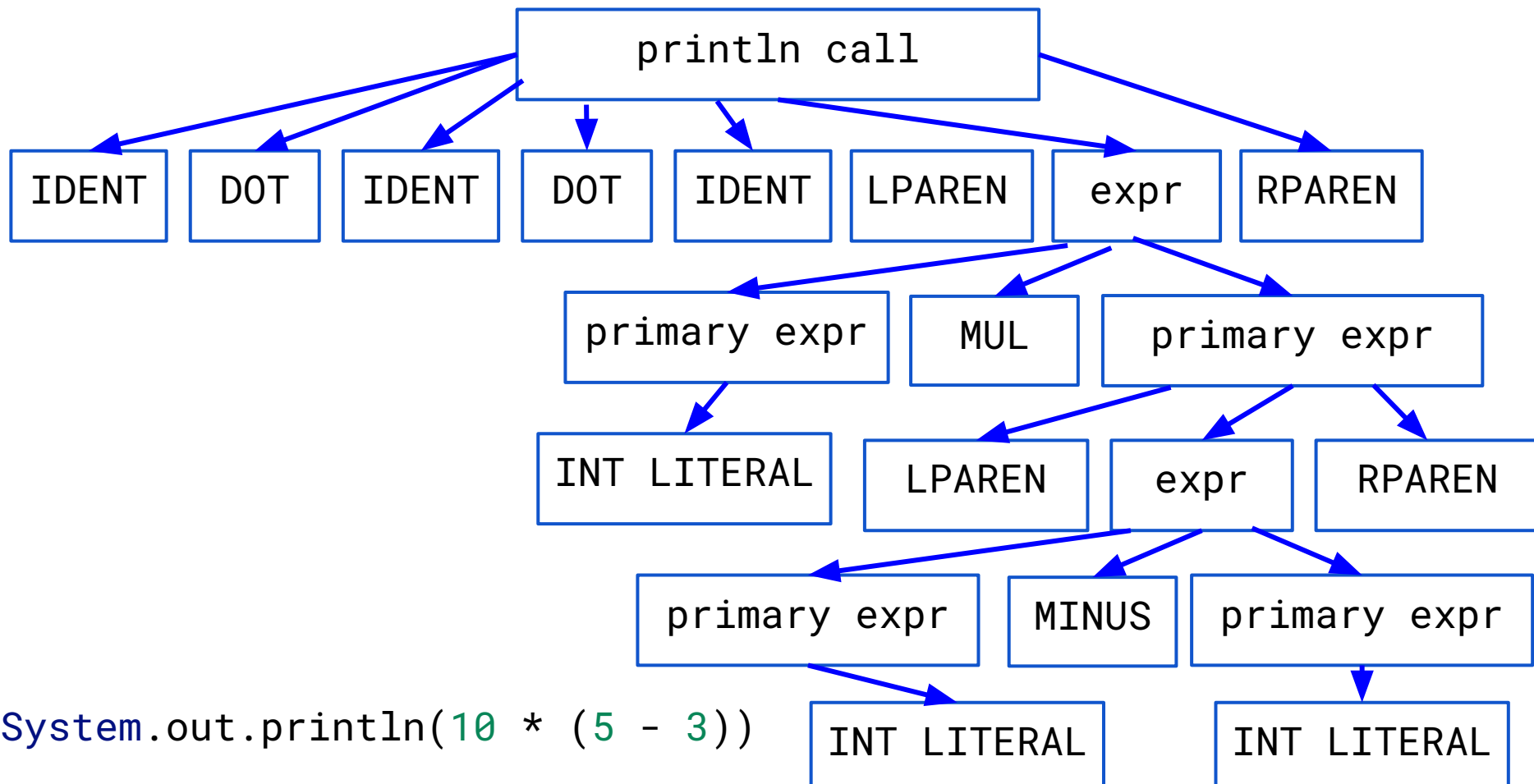
Abstract Syntax Tree

Абстрактное синтаксическое дерево — AST

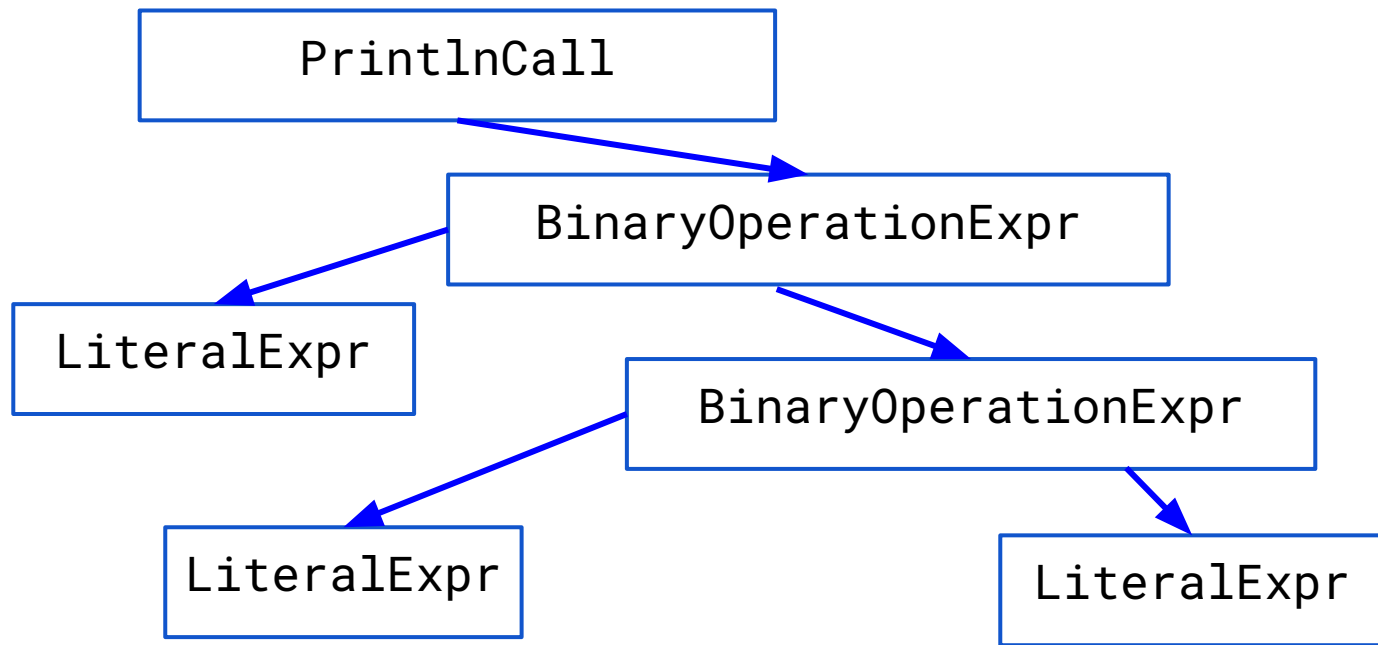
1. N-арное дерево
2. Узел соответствует нетерминалу
3. Узел представлен:
 - Подклассом — LiteralExpr, BinaryOpExpr, MethodCallExpr
 - Свойствами подкласса

```
System.out.println(10 * (5 - 3))
```

Дерево разбора (Parse Tree)



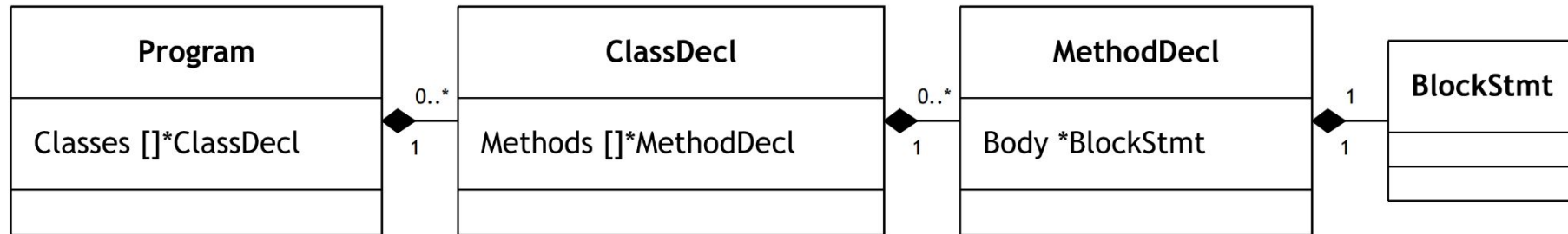
Абстрактное синтаксическое дерево



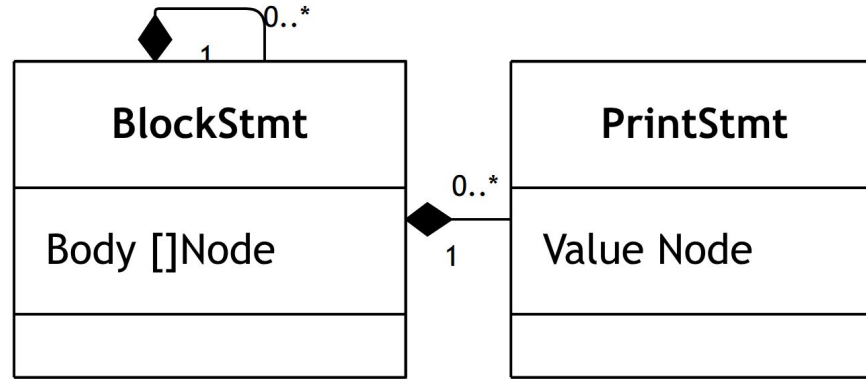
`System.out.println(10 * (5 - 3))`

AST языка MiniJava

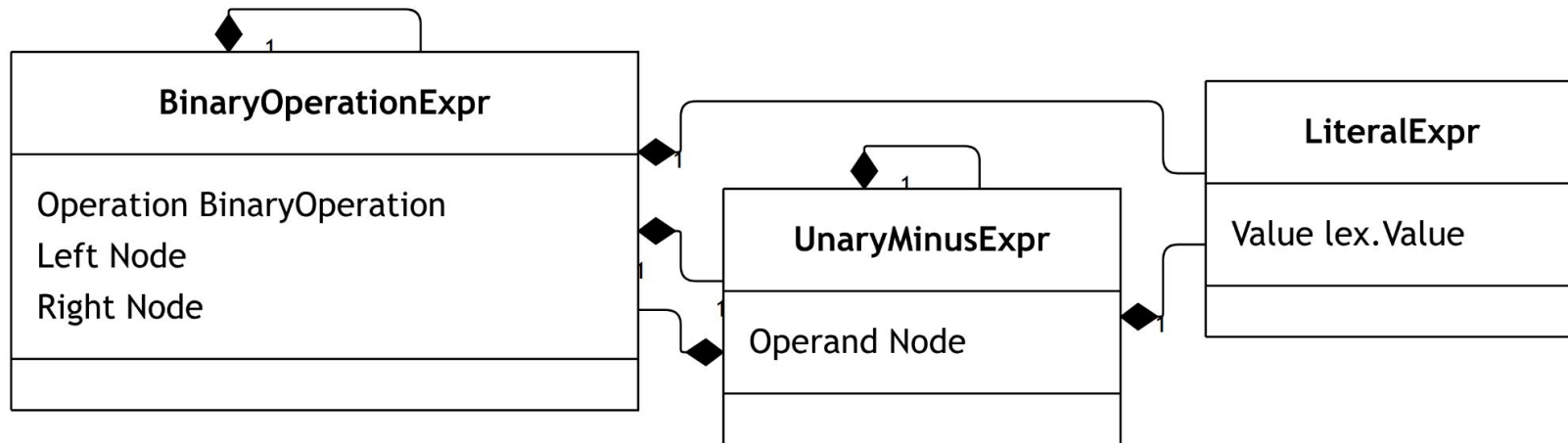
Объявления — declarations



Инструкции — statements



Выражения — expressions



Обработка построенного AST

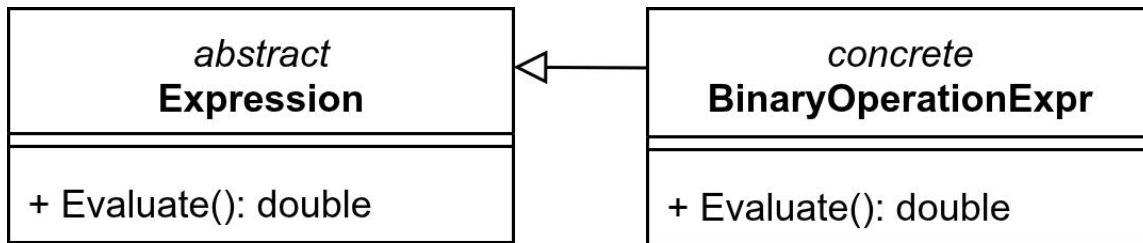
Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	Дерево: • Parse Tree • AST — Abstract Syntax Tree
AST нужен для разных целей, сложность не должна расти	???

Диспетчеризация

1. Диспетчеризация — действие выбирается в зависимости от подтипа объекта

- Вызываем метод Expression.Evaluate()
- → вызывается метод BinaryOperationExpr.Evaluate()



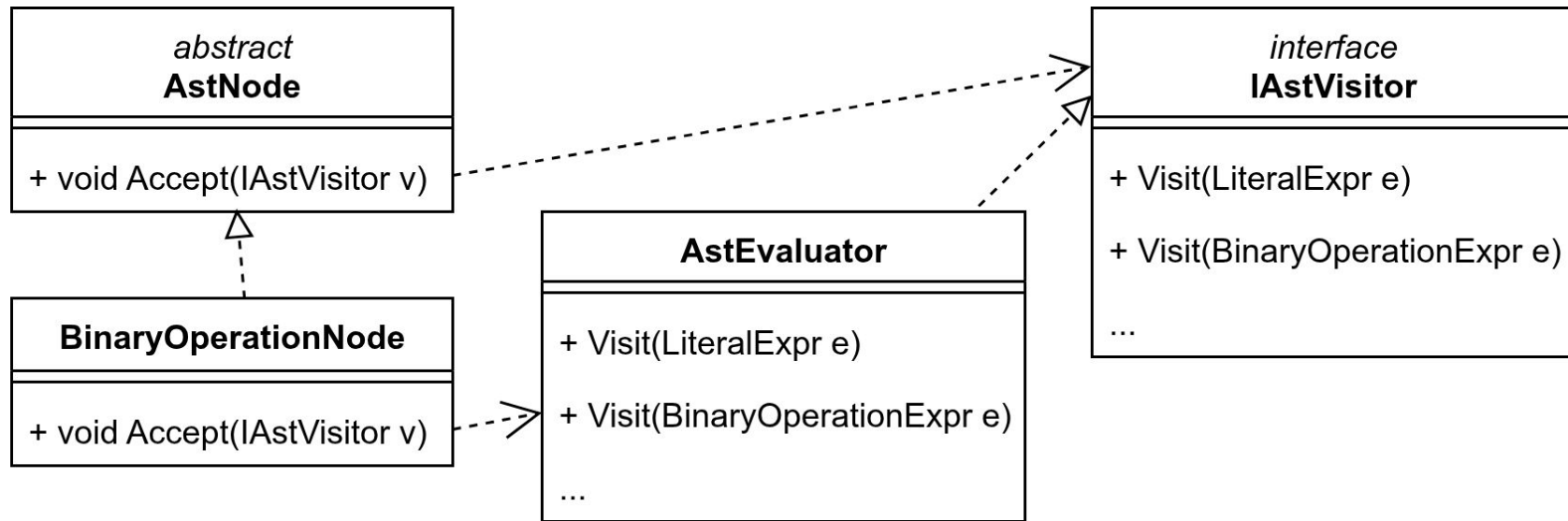
Диспетчеризация

1. Диспетчеризация — действие выбирается в зависимости от подтипа объекта
2. Двойная диспетчеризация — действие выбирается в зависимости от подтипа объекта **и аргумента**
 - Шаблон «Посетитель» (Visitor)
 - Сопоставление (Pattern Matching) по подтипам

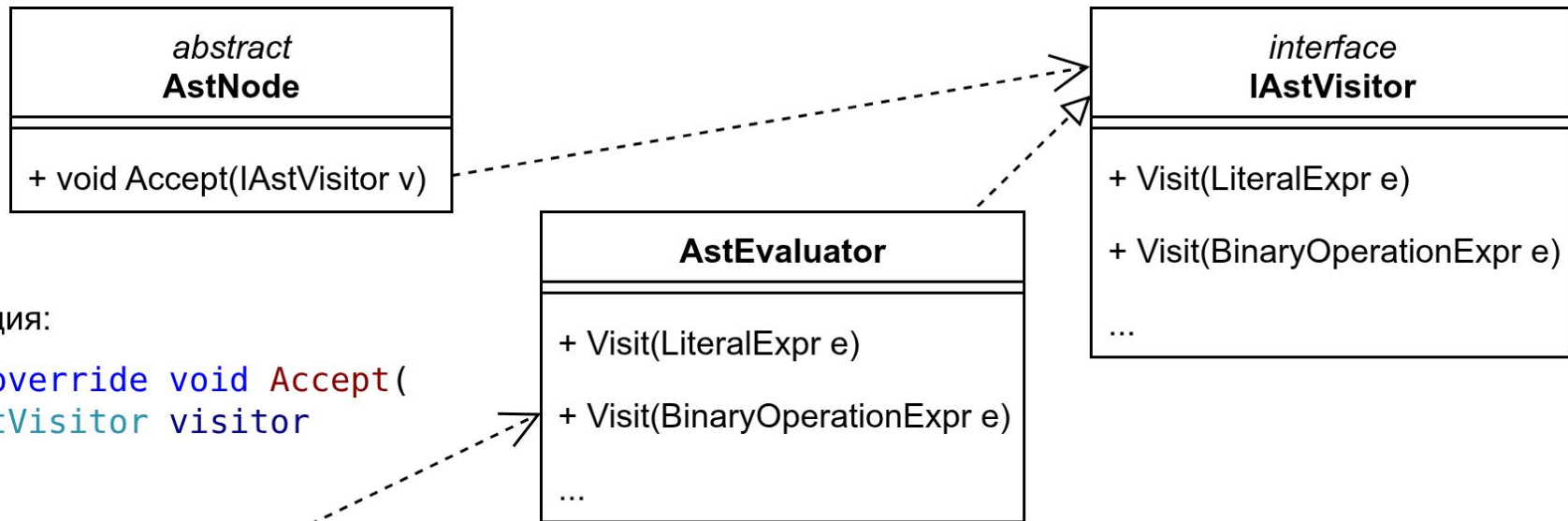
Сопоставление по типу (Pattern Matching by type)

```
return _value switch  
{  
    string s => s,  
    double d => d.ToString(CultureInfo.InvariantCulture),  
    _ => throw new NotImplementedException(),  
};
```

Шаблон «Посетитель» (Visitor)



Шаблон «Посетитель» (Visitor)



Реализация:

```
public override void Accept(  
    IAstVisitor visitor  
)  
{  
    visitor.Visit(this);  
}
```

Вычисление выражений по AST

Вычисление выражения по AST

1. Создаём стек
2. Обходим дерево
3. Возвращаем единственное оставшееся значение в стеке

```
public double Evaluate(AstNode node)
{
    node.Accept(this);
    // TODO: бросить исключение, если в стеке не 1 значение
    return _values.Pop();
}
```

Литерал

```
public void Visit(LiteralExpression e)
{
    _values.Push(e.Value);
}
```

Чтение переменной или константы

```
public void Visit(VariableExpression e)
{
    _values.Push(_context.GetValue(e.Name));
}
```

Изменение значения переменной

```
public void Visit(AssignmentExpression e)
{
    // NOTE: Вычисляем выражение, и затем присваиваем
    // его значение переменной, сохраняя результат в стеке.
    e.Value.Accept(this);
    double value = _values.Peek();
    _context.AssignVariable(e.Name, value);
}
```

Унарная операция

```
public void Visit(UnaryOperationExpression e) {  
    e.Operand.Accept(this);  
    switch (e.Operation) {  
        case UnaryOperation.Minus:  
            _values.Push(-_values.Pop());  
            break;  
        case UnaryOperation.Plus:  
            break;  
        default:  
            throw new NotImplementedException(...);  
    }  
}
```

Бинарная операция

```
public void Visit(BinaryOperationExpression e) {  
    e.Left.Accept(this);  
    e.Right.Accept(this);  
    double right = _values.Pop()  
    double left = _values.Pop();  
    switch (e.Operation) {  
        case BinaryOperation.Plus:  
            _values.Push(left + right);  
            break;  
        // ...  
        default:  
            throw new NotImplementedException(...);  
    }  
}
```

Проблемы / решения

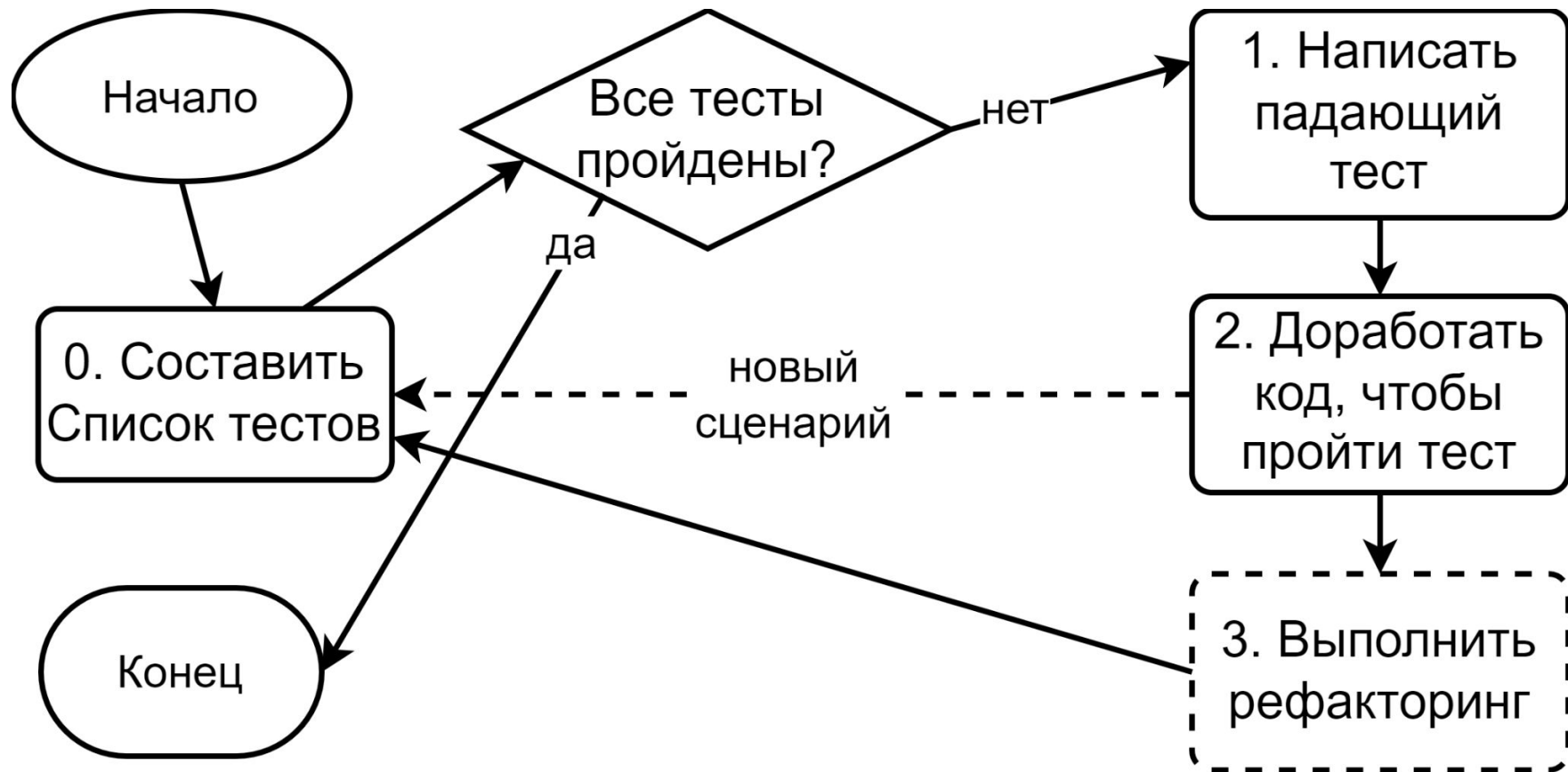
Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	Дерево: • Parse Tree • AST — Abstract Syntax Tree
AST нужен для разных целей, сложность не должна расти	Используем паттерн Visitor

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	Дерево: • Parse Tree • AST — Abstract Syntax Tree
AST нужен для разных целей, сложность не должна расти	Используем паттерн Visitor
Рефакторинг может сломать программу	???

Модульные тесты

Каноничный TDD



Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. [Canon TDD](#)

Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. [Canon TDD](#)

Вопрос: какая цель важнее всего?

Цели модульного тестирования

1. Защита от регрессий
2. Тесты как документация модуля
3. Уверенность в каждом модуле
4. Тесты как способ проектирования — см. [Canon TDD](#)

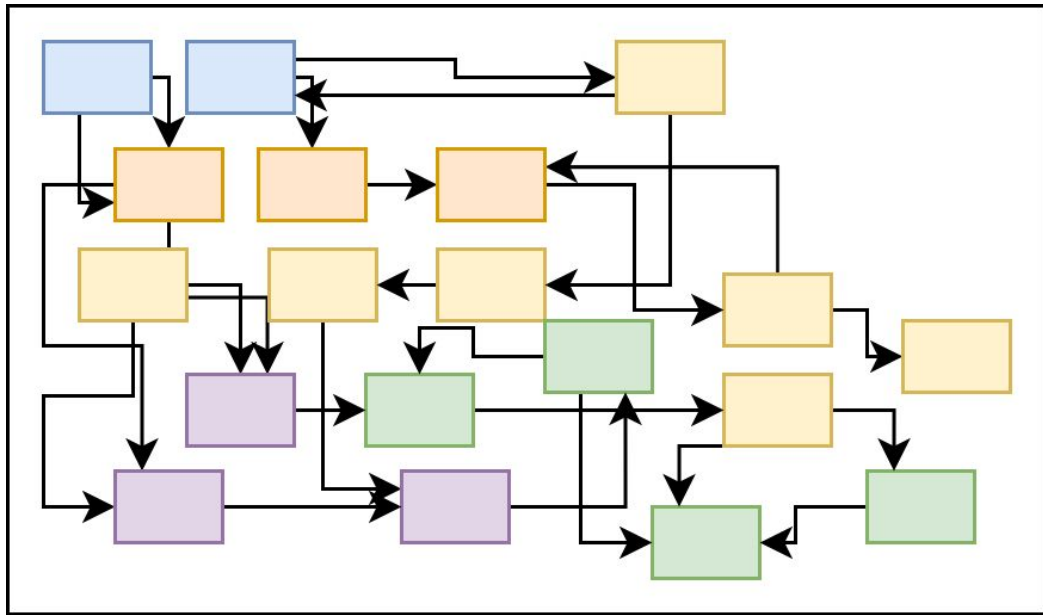
Вопрос: какая цель важнее всего?

Ответ: цели упорядочены по **возрастанию** важности

Разработка системы по TDD

Стиль ООП

Программа — это набор
взаимодействующих
объектов

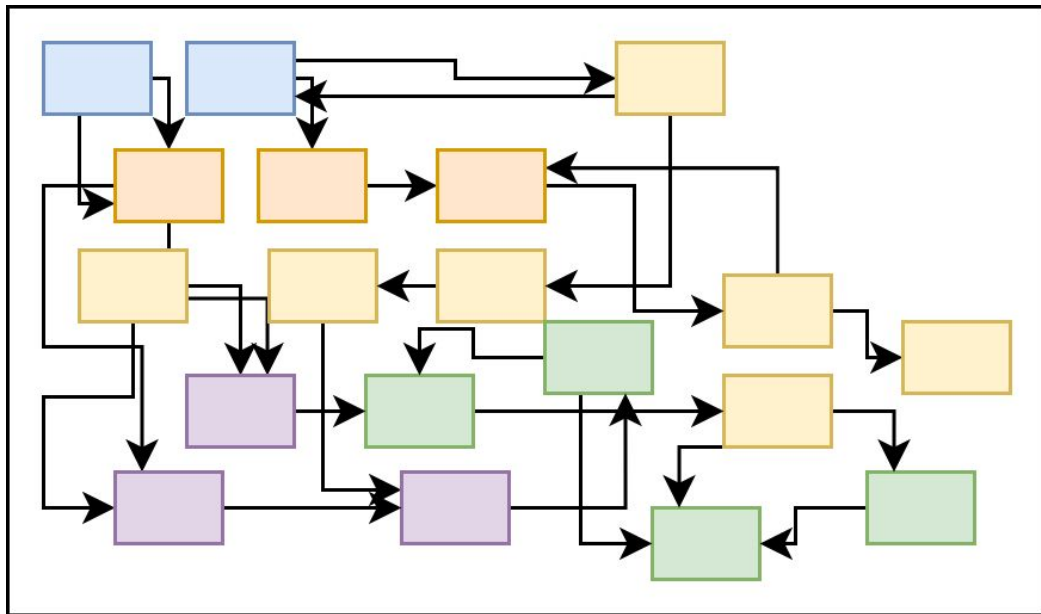


Разработка системы по TDD

1. Делим программу на модули
2. Каждый модуль пишем по TDD

Стиль ООП

Программа — это набор взаимодействующих объектов



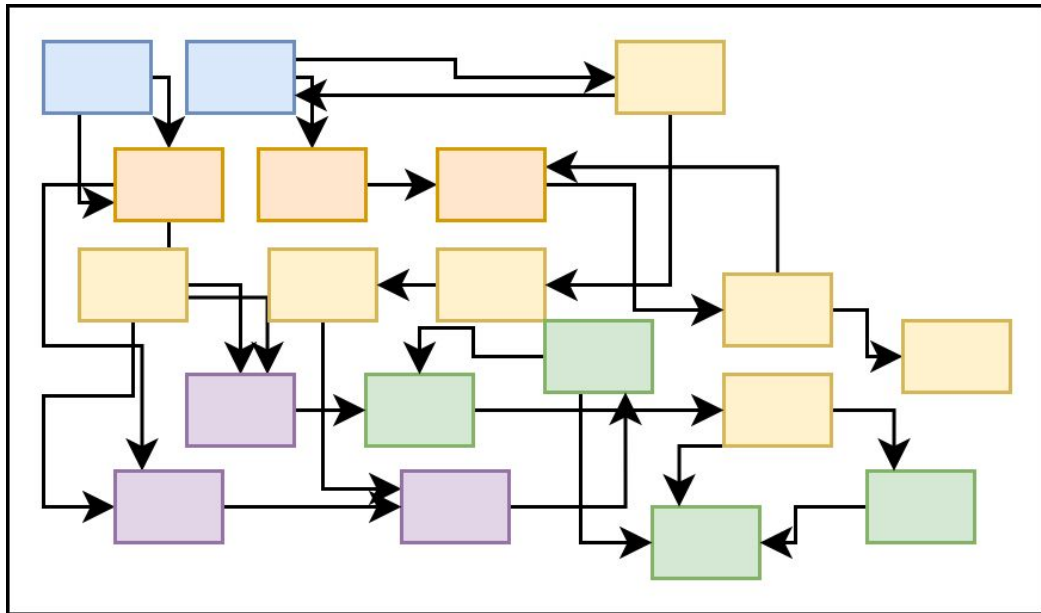
Разработка системы по TDD

1. Делим программу на модули
2. Каждый модуль пишем по TDD

1. UML Class
2. UML Sequence

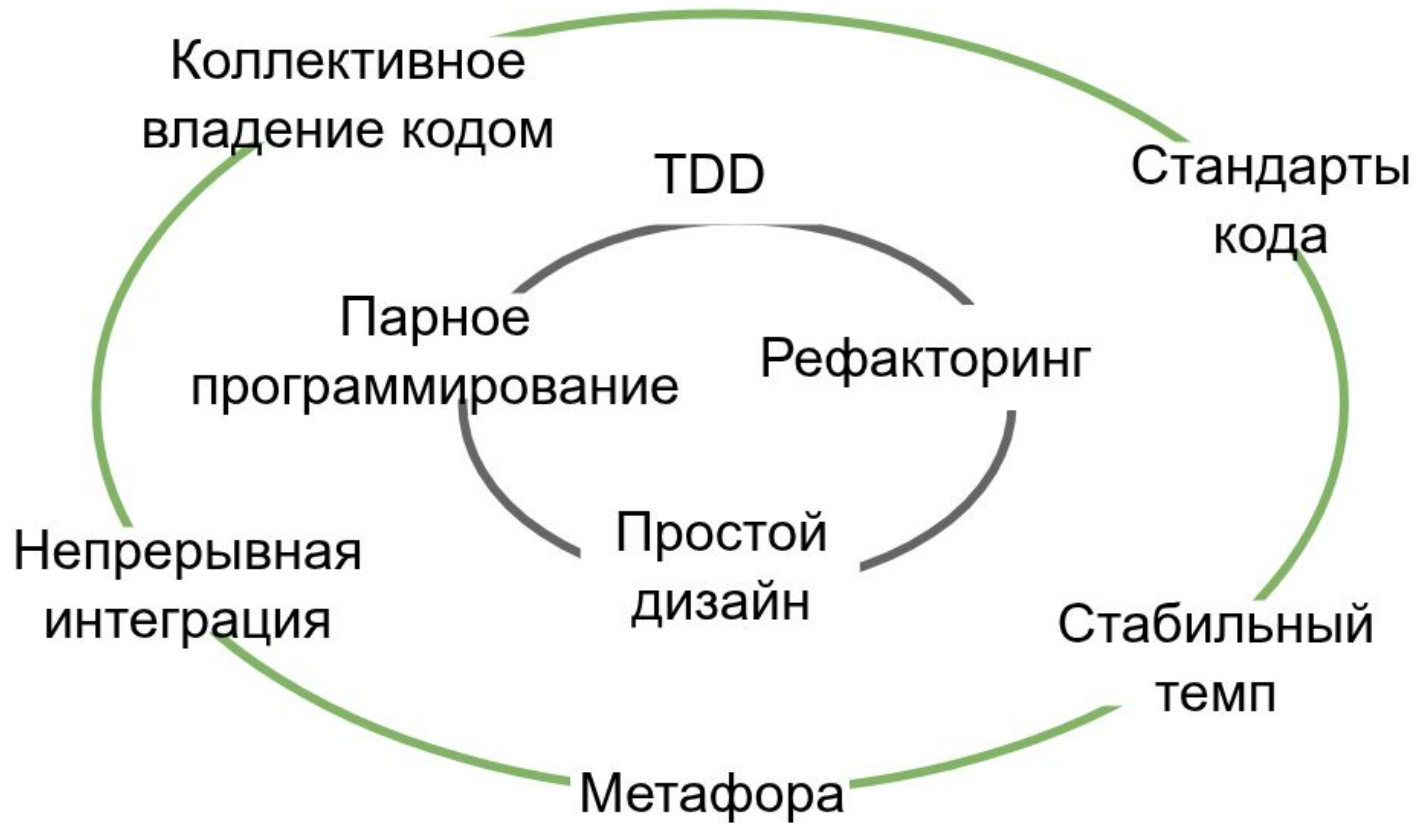
Стиль ООП

Программа — это набор взаимодействующих объектов

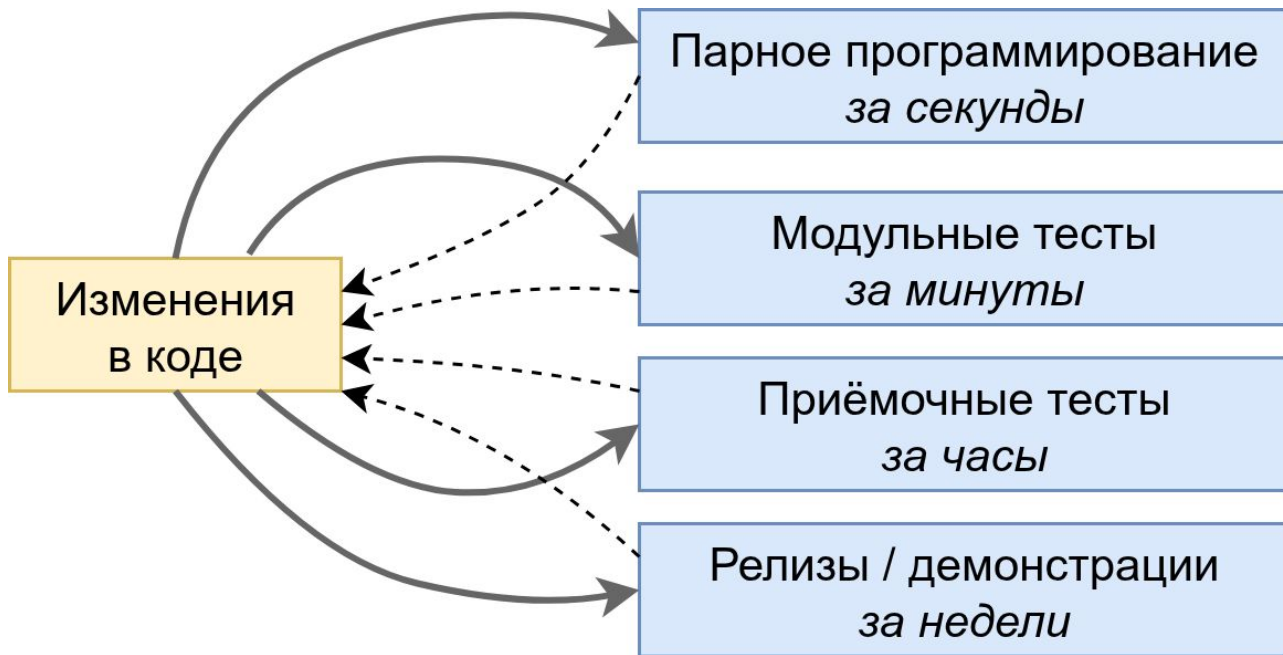


Практики eXtreme Programming

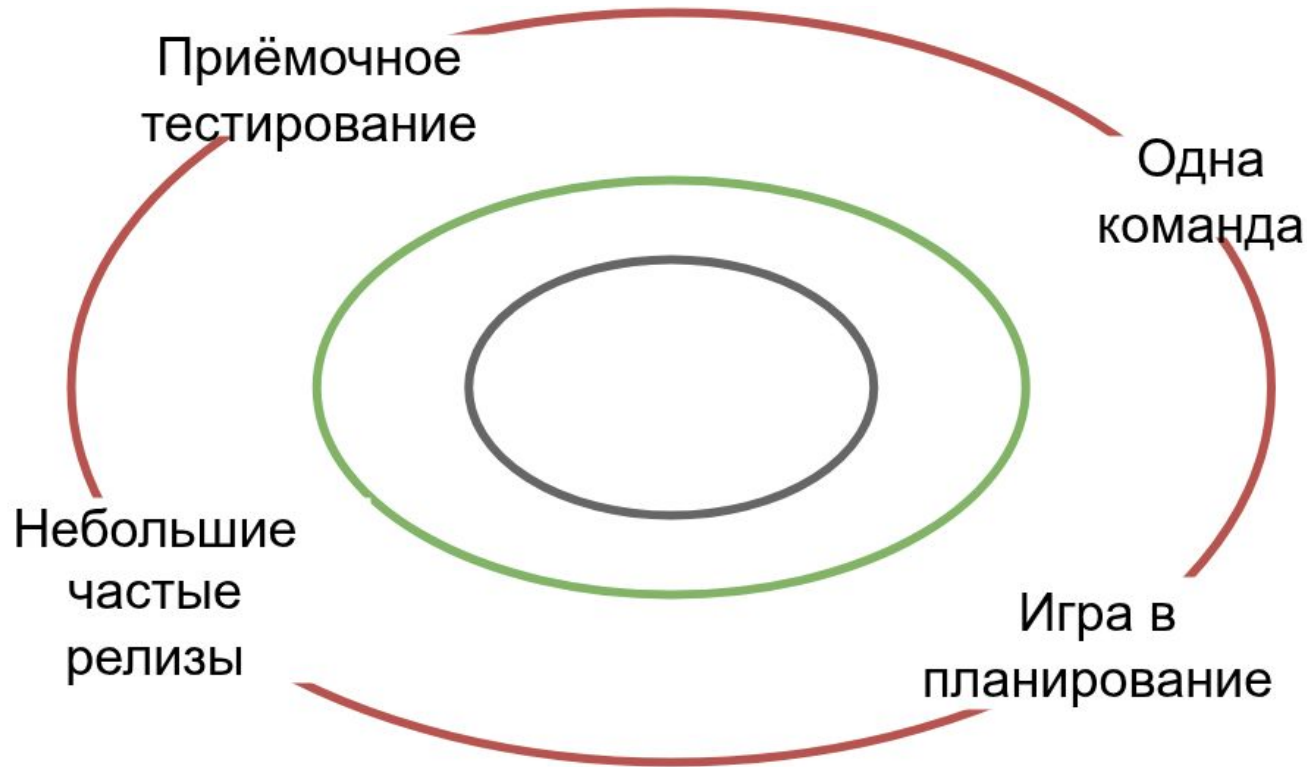
Практики eXtreme Programming — два круга из трёх



Петли обратной связи в Agile (feedback loops)



Практики eXtreme Programming — третий круг



Приёмочные тесты

Функциональное тестирование

Принципы Structural Testing (1970-е):

- Тестирование чёрного ящика
 - Тестируют наблюдаемое извне поведение
 - Пример: ввод/вывод консольной программы
 - Это **функциональные** тесты
- Тестирование прозрачного (белого) ящика
 - Тестируют внутренние интерфейсы программы

Интеграционные тесты

1. Проверяют слои и модули совместно
2. Чем больше охват слоёв и модулей — тем лучше
3. Чем меньше тестовых дублёров — тем лучше
 - Нет подмены **внутренних** зависимостей
 - Нет подмены **внешних управляемых** зависимостей

Приёмочные тесты

Проверяют, соответствует ли ПО решаемой задаче:

- Customer Tests — из eXtreme Programming, 1998
- Acceptance Tests — из Agile, 2000-е

Проблемы / решения

Проблема	Решение
Парсинг нужен для разных целей, его сложность растёт	Продукция парсера — логическая модель программы
Как представить логическую модель программы?	Дерево: • Parse Tree • AST — Abstract Syntax Tree
AST нужен для разных целей, сложность не должна расти	Используем паттерн Visitor
Рефакторинг может сломать программу	Приёмочные интеграционные функциональные тесты

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
 - Они устойчивы к изменению архитектуры
2. ???
3. ???

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
 - Они устойчивы к изменению архитектуры
2. Проводим рефакторинг
 - Добавляем AST
3. ???

Трёхэтапный процесс рефакторинга

1. Покрываем код приёмочными тестами
 - Они устойчивы к изменению архитектуры
2. Проводим рефакторинг
 - Добавляем AST
3. Реализуем новую функциональность
 - Ветвления, циклы, функции

Спасибо за внимание!
Вопросы?