Выделение типов в универсальном классовом представлении для статического анализа исходного кода

Пустыгин А.Н.
Зубов М.В.
Старцев Е.В.

Челябинский Государственный Университет
2013

Общая схема изучения ПО по исходному тексту с помощью промежуточных представлений

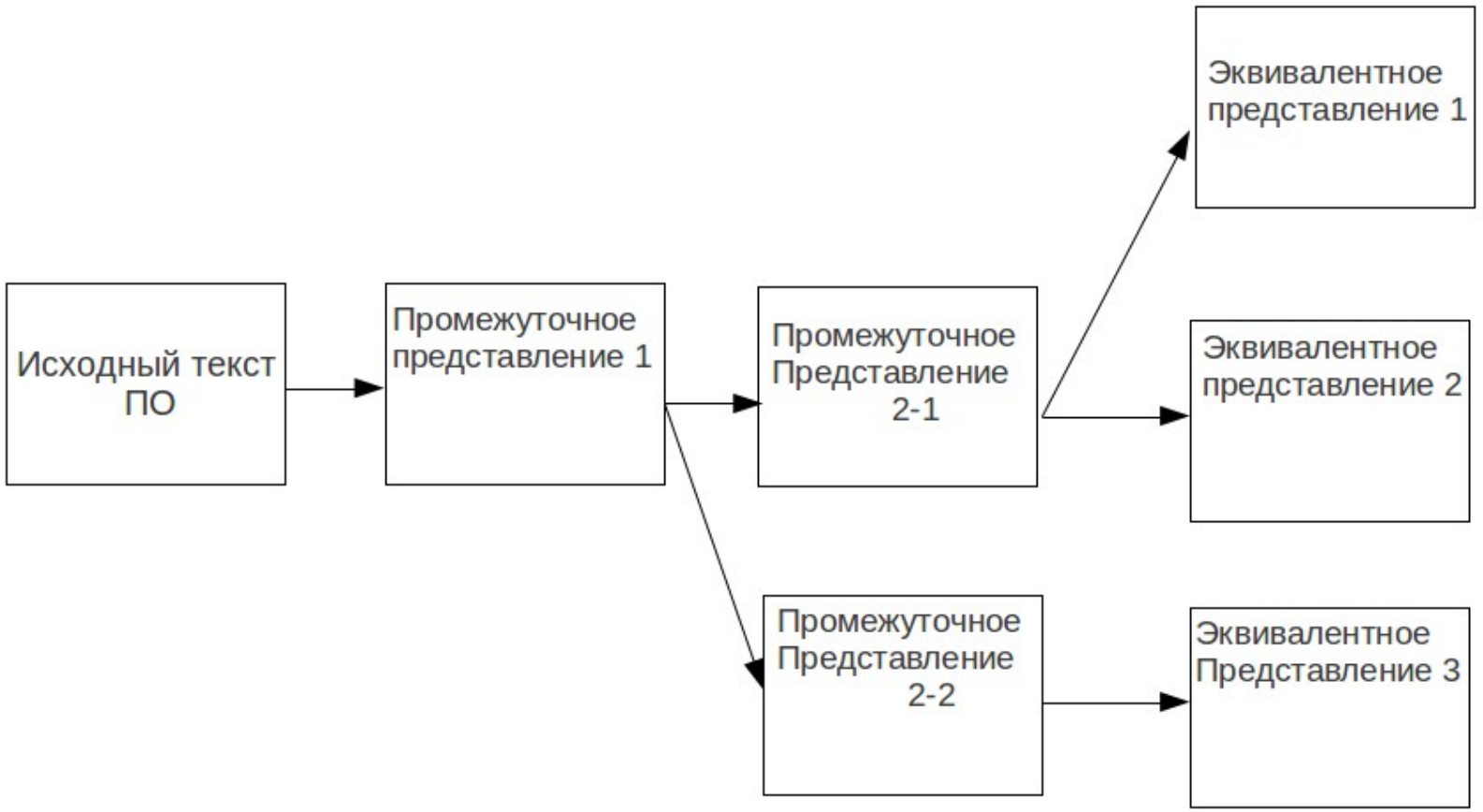
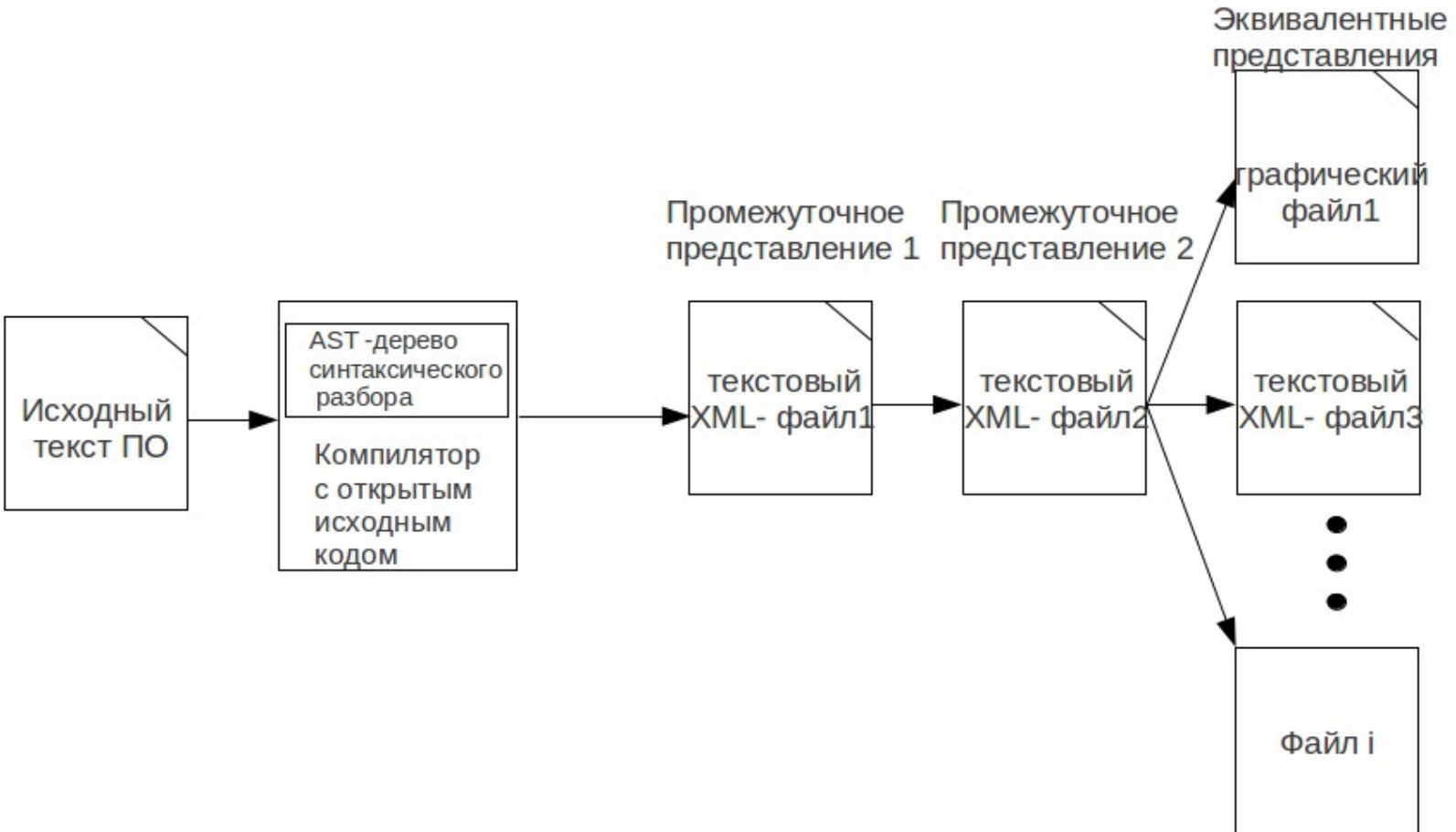


Схема преобразования информации при построении эквивалентных представлений исходного текста



Промежуточные представления исходного текста

набор данных оговоренного формата, предназначенный для последующего построения эквивалентных представлений исходного текста

Эквивалентные представления исходного текста

набор данных оговоренного формата, предназначенный для последующего анализа исходного текста по тому или иному критерию

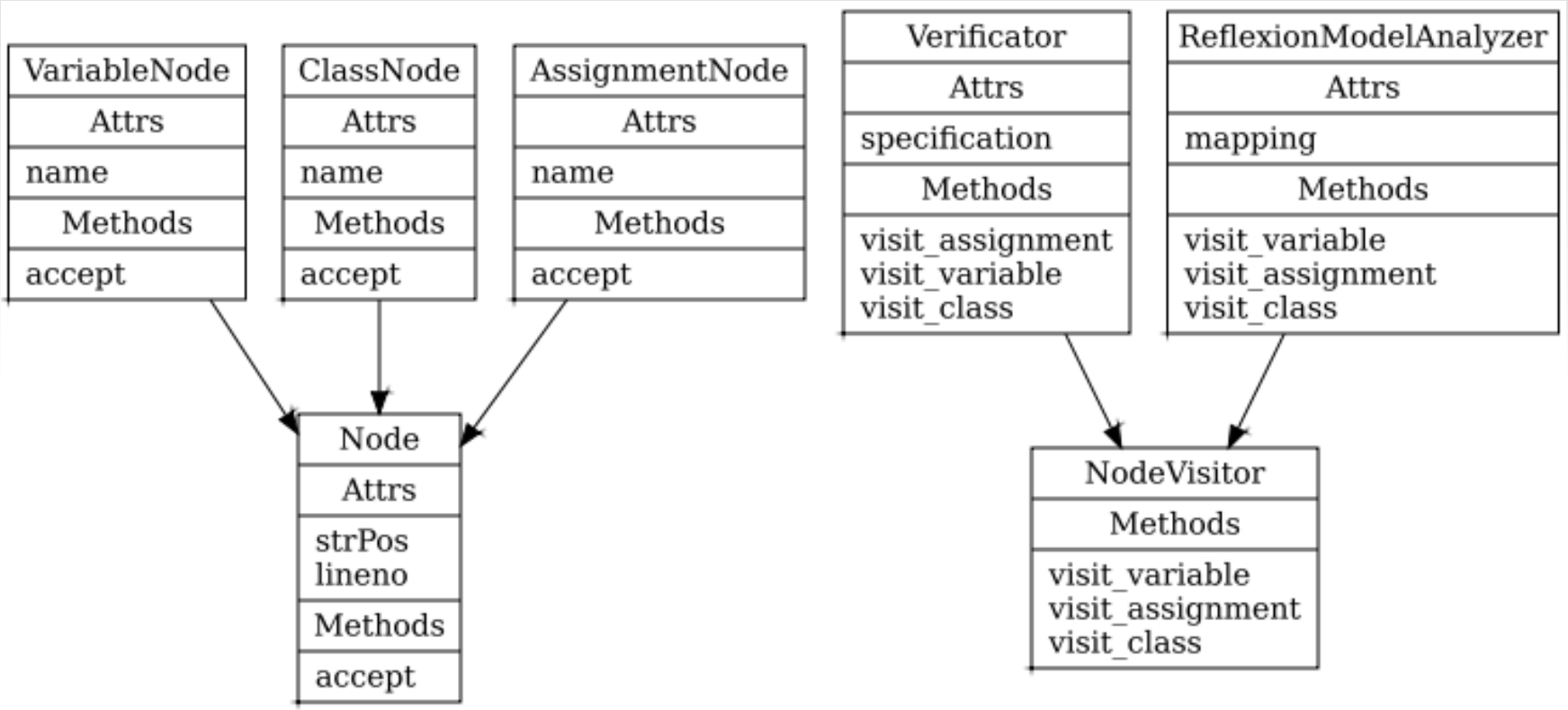
Универсальное классовое промежуточное представление

Одно из промежуточных представлений, отражающее диаграмму классов проекта, предназначенное для исходных текстов на нескольких языках (Python, Java, C++). Используется только уровень структуры классов

Типы данных в промежуточном классовом представлении

- Типы полей
- Типы возвращаемых из методов значений
- Типы аргументов методов
- включены только «*внутренние типы*» проекта

Визуализация диаграммы классов для двух одинаковых простых проектов JAVA и Python на основе универсального классового проедставления



Обработка эквивалентных представлений на примере выделения суперкласса на диаграмме классов

Суперкласс — вершина иерархии наследования классов проекта на основе общего множества методов для групп объектов, которые могут быть вынесены в общий предок (Вид рефакторинга, выполняемый по именам и количеству параметров методов)

Внутренние и внешние классовые типы

Внутренние — определены внутри исследуемого проекта
Внешние — не определены в проекте:

- системные
- библиотечные

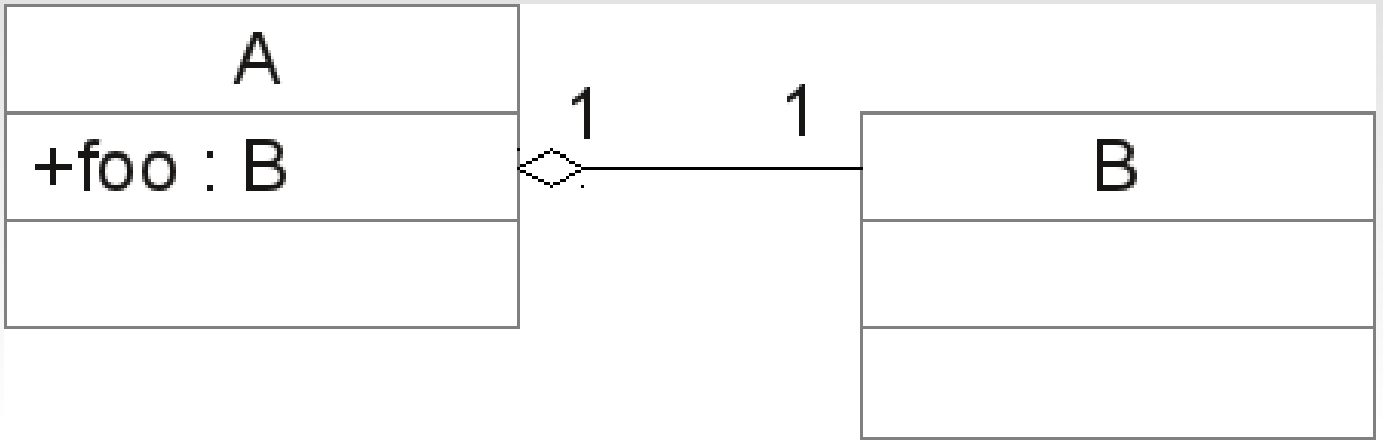
Примеры внешних типов

- Массивы
- Встроенные типы (особенно коллекции)
- Системные и библиотечные типы

Два типа связей агрегации в диаграмме классов

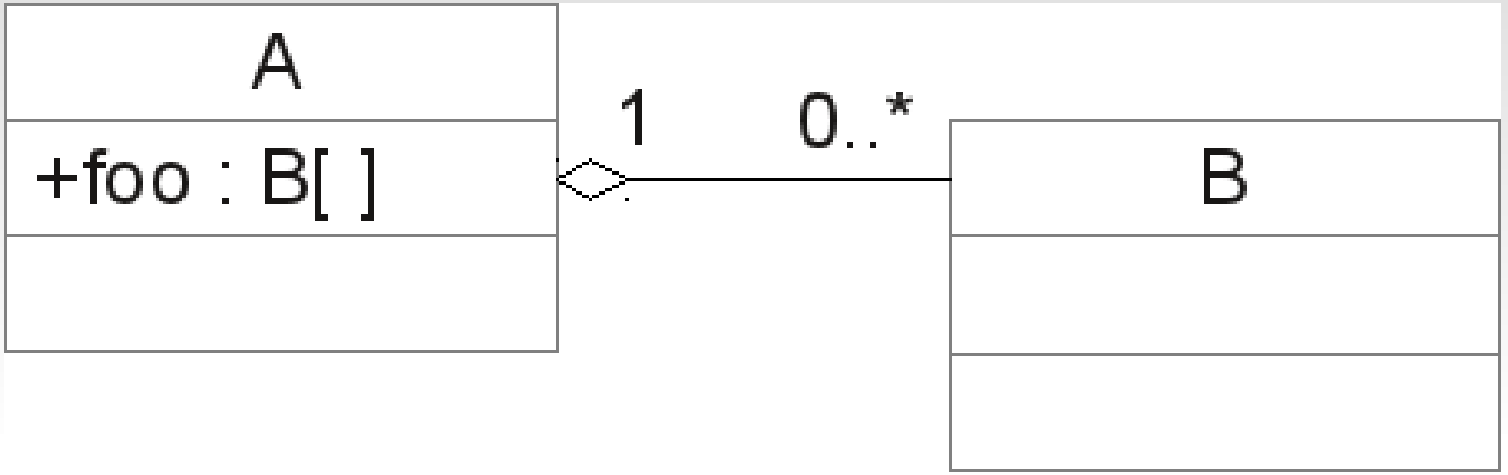
- Простой тип отражает связь «один-к-одному»
- массивы и коллекции отражают связь «многие-к-одному»

Один-к-одному




```
class A{
   private B foo;
}
```

Многие-к-одному




```
class A{
   private B[] foo;
}
```

Языки со статической типизацией

- Java, C++
- Имя типа можно определить через AST
- Типы легко сопоставить с известными *внутренними*

Языки с динамической типизацией(Python)

Тип объекта в исходном тексте не определен, он становится известен только на момент исполнения.

Для построения статической диаграммы классов требуется специальный алгоритм выделения типов

Сложности анализа структуры классов произвольного типа

- *Внешние* типы не рассматриваются при анализе, но внешние типы могут содержать в себе внутренние, поэтому возникают транзитивные связи, которые теряются

Результаты анализа Java-проектов.

Проект	Всего методов	Проблемных методов
Javac	2076	397
FindBugs	5538	787
SquirreL	7798	1324

Проблемными методами названы те, чьи имена совпадают в разных классах, но не унаследованы ими от общего суперкласса.

Результаты анализа Python-проектов.

Проект	Всего методов	Проблемных методов
Django	2079	334
Twisted	8651	1291
Logilab	491	110

Выделение типов в универсальном классовом представлении для динамических языков

Утиный тест (*duck test*)

- Предмет можно идентифицировать по его внешним признакам:
  - Это выглядит как утка,
  - Это плавает как утка,
  - Это крякает как утка,
  - Значит, вероятно, это утка!*

*«*If it looks like a duck, swims like a duck and quacks like a duck, then it probably is a duck*»

Пример утиной типизации в Phyton

```
from my_animals import MyRabbit
import dr_frankenstein
import random

class Zoo(object):
    _rabbit = None
    _duck = None
    _lemmings = None

    def __init__(self):
        self._rabbit = MyRabbit()
        self._duck = dr_frankenstein.get_duck()
        self._lemmings = [dr_frankenstein.get_lemming() for x in range(10000)]

    def life(self):
        while 1:
            self._rabbit.eat_carrot()
            self._duck.move_to_pool()
            self._duck.swim()
            if(self._duck.excitement):
                self._duck.quack()
            if(len(self._lemmings) > 0):
                current = random.randint(0,len(self._lemmings)-1)
                self._lemmings[current].suicide()
                del self._lemmings[current]
```

Поле _rabbit

- Класс хранимого объекта очевиден, так как инициализируется в явном виде

```
...
self._rabbit = MyRabbit()
...
```

Поле _duck

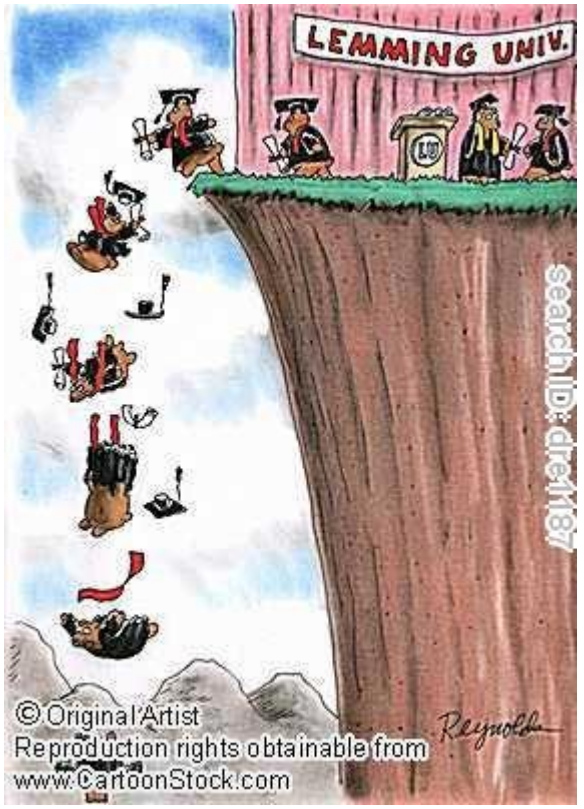
- О классе объекта поля _duck можно судить только из его поведения («утиный тип»)
- Он может быть даже не «виден» в области видимости класса Zoo
- Классу Zoo «не интересно», кто на самом деле эта «утка»

```
...
self._duck = dr_frankenstein.get_duck()
...
self._duck.move_to_pool()
self._duck.swim()
if(self._duck.excitement):
        self._duck.quack()
...
```

Поле _lemmings

- В поле хранится список элементов (связь «многие к одному»)
- О классе объектов-элементов списка можно судить только исходя из их поведения(«утиный тип»)

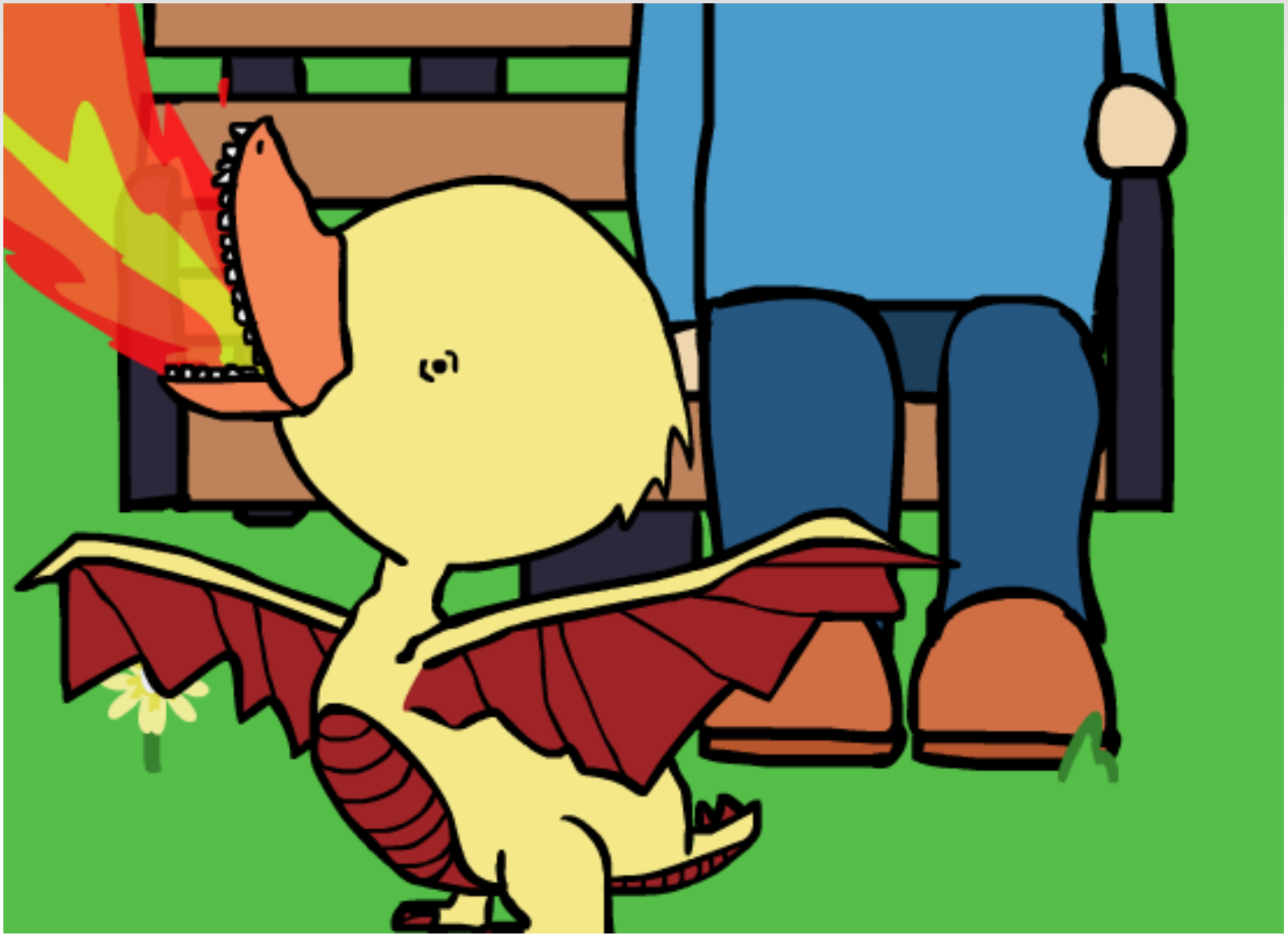
```
...
self._lemmings = [dr_frankenstein.get_lemming() for x in range(10000)]
...
if(len(self._lemmings) > 0):
        current = random.randint(0,len(self._lemmings)-1)
        self._lemmings[current].suicide()
        del self._lemmings[current]
...
```



А на самом деле утка не так проста, как кажется!

```
модуль dr_frankenstein:

from my_animals import MyLemming, CleverDragon

def get_lemming():
        return MyLemming()

def get_duck();
        return CleverDragon()
...

класс CleverDragon:

class CleverDragon(object):

    excitement = True

    def swim(self):
        ...
    def move_to_pool(self):
        ...
    ...
    def _fiery_breath(self):
        ...

    def quack(self):
        self._fiery_breath()
```

Утка!



Детали реализации прототипа инструмента

Для Python использовался генератор на основе библиотеки logilab-astng, что позволило сразу получать информацию о классах проекта
Для Java использовался генератор, использующий AST. Дерево разбора получалось из XML-документа, генерируемого препаратом компилятора Open JDK
Программно-аппаратная платформа для выполнения тестов: AMD Phenom(tm) II X4 955, 8GB RAM, OC-Archlinux

Результаты применения прототипа при анализе эквивалентных представлений

Проект	Всего «уток»	«Пустые утки»	«Найден-ные утки»	«Утки сложного типа»	Всего атрибутов в полях классов проекта	Время построения представления
Twisted 12.0.0	4501	2335	1767	616	8644	2м. 33с.
Logilab-astng 0.23.1 + Logilab-common 0.58.0	230	132	63	64	597	6с.
Django 1.4.1	1125	544	393	307	3490	41с.

Терминология в таблице

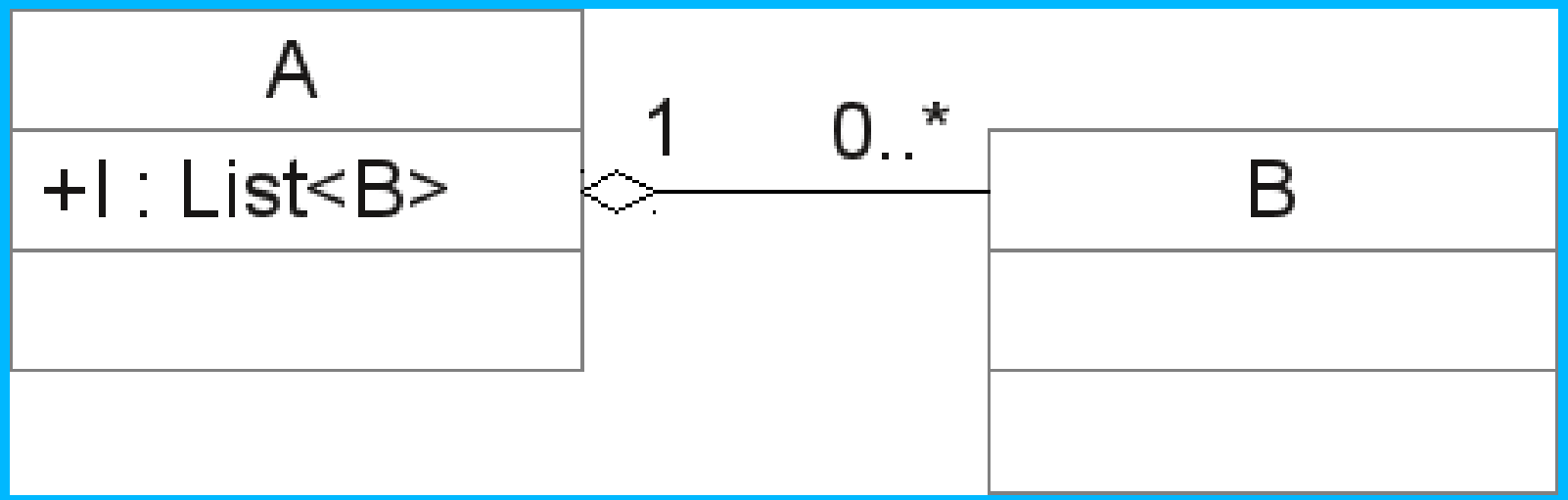
- «Утки» – поля классов, о которых удалось извлечь информацию на основе обращений
- «Пустые утки» – те из «уток», о полях и методах которых не удалось извлечь никакой информации (кандидаты не искались)
- «Найденные утки» – те из «уток», для которых удалось установить хотя бы 1 класс-кандидат
- «Утки» сложного типа, «утки» для которых удалось установить связь типа «1 ко многим», примером служит уже рассмотренное поле _lemmings класса Zoo

Шаблонные конструкции языков со статической типизацией

- Можно выделить 2 группы
- 1 группа (внешний тип):
  - `List<Object>`
  - `List<B>`
- 2 группа (внутренний тип):
  - `A<Object>`
  - `A<B>`

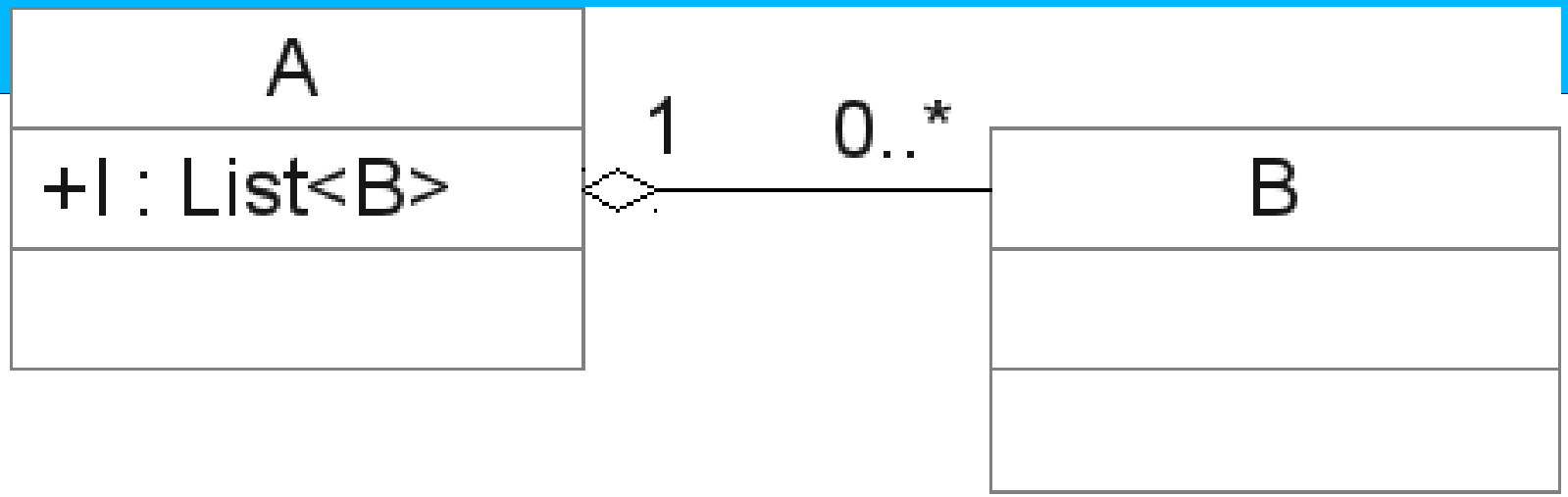
Первая группа

- List<Object> — не содержится проектных классов, будет отброшена при анализе
- List<B> — содержит проектный тип. Коллекция элементов B.



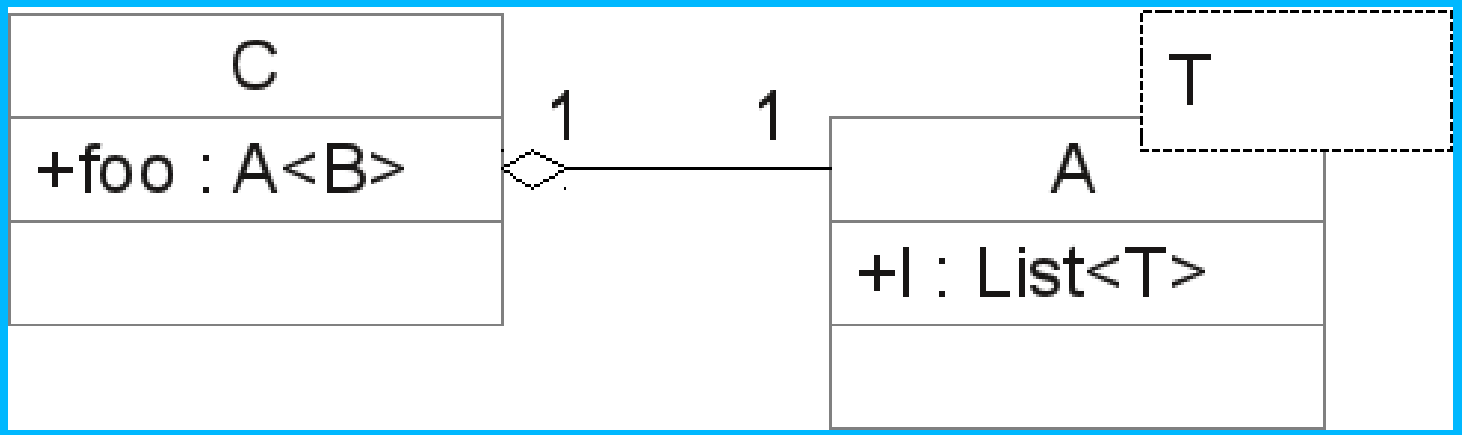
Первая группа

```
class A{
   public List<B> I;
}
```



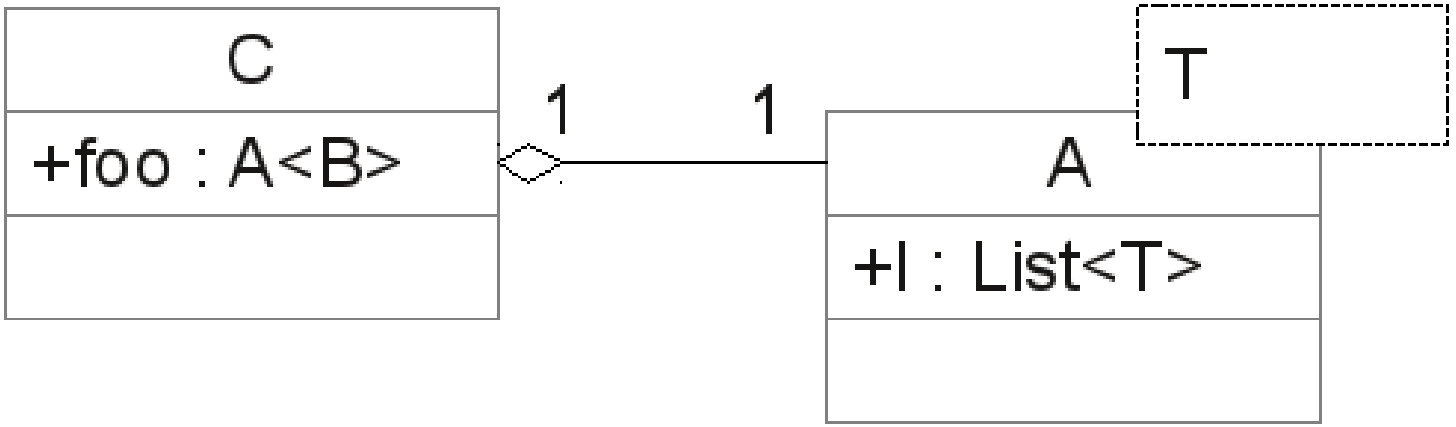
Вторая группа

- A<Object>, A<B> — шаблон влияет на внутреннюю структуру класса A
- Класс A описан в проекте, потому доступно его устройство
- Связь 1-к-1 с проектным классом



Вторая группа

```
class C{
   public A<B> foo;
}
class A<T>{
   public List<T> I;
}
```



Общие свойства нового представления

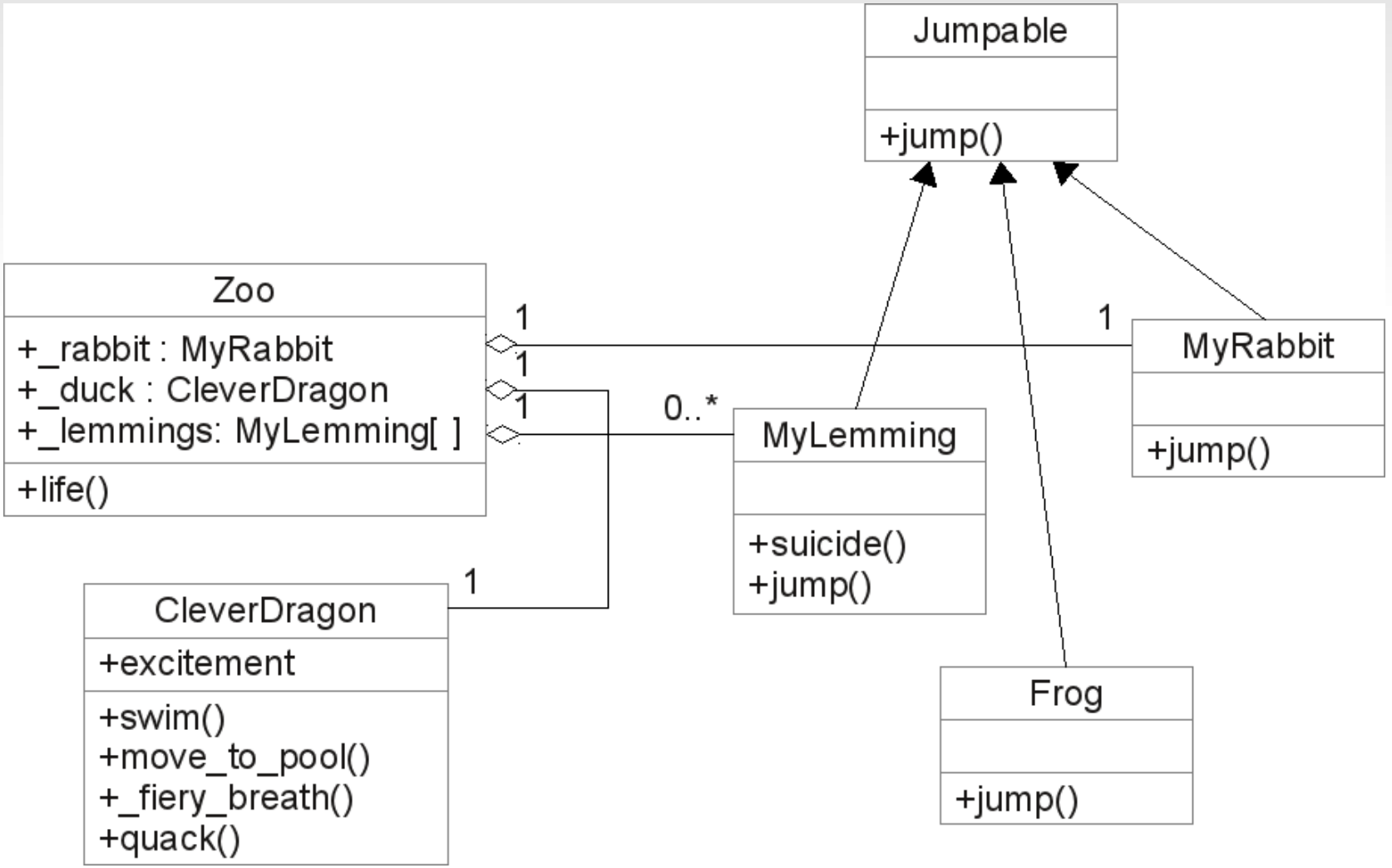
- Содержит 2 типа связей между типами
- Позволяет выделять 3 типа ассоциаций между классами
- Содержит информацию о шаблонах

Пример использования представления.Исходный текст Python

```
class Zoo(object):
    _rabbit = None
    _duck = None
    _lemmings = None
    def _init_(self):
        self._rabbit = MyRabbit()
        self._duck = dr_frankenstein.get_duck()
        self._lemmings = [dr_frankenstein.get_lemming()
                          for x in range(10000)]
    def life(self):
        while 1:
            self._rabbit.eat_carrot()
            self._duck.move_to_pool()
            self._duck.swim()
            if (self._duck.excitement):
                self._duck.quack()
            if (len(self._lemmings) > 0):
                current = random.randint(0,
                          len(self._lemmings)-1)
                self._lemmings[current].suicide()
                del self._lemmings[current]
```

```
class Jumpable(object):
    def jump(self):
        pass

class MyRabbit(Jumpable):
    def eat_carrot(self):
        ...
    def jump(self):
        ...

class Frog(Jumpable):
    def jump(self):
        ...

class CleverDragon(object):
    excitement = True
    def swim(self):
        ...
    def move_to_pool(self):
        ...
    def _fiery_breath(self):
        ...
    def quack(self):
        self._fiery_breath()

class MyLemming(Jumpable):
    def suicide(self):
        ...
    def jump(self):
        ...
```

Пример использования представления. Диаграмма классов проекта

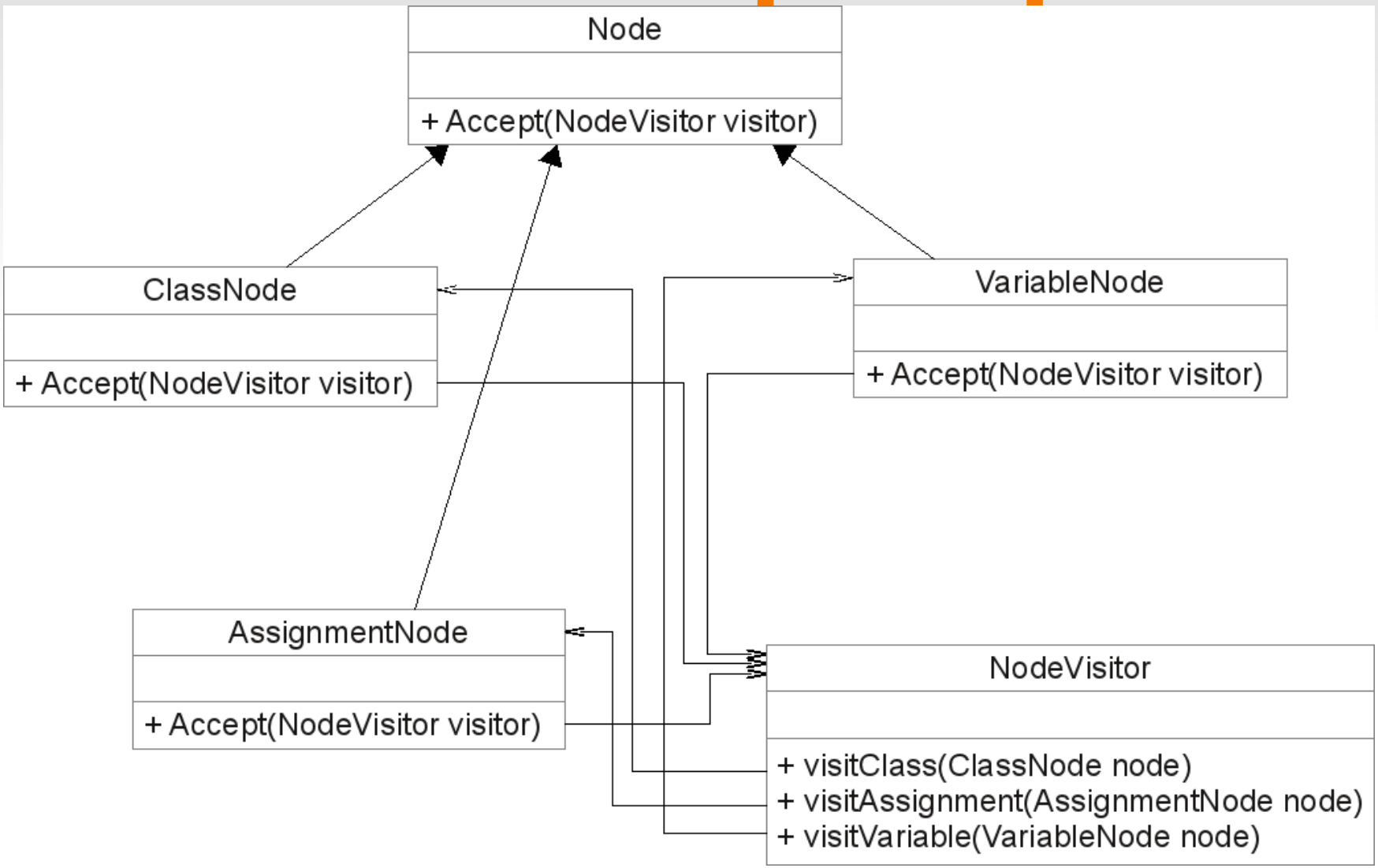


Пример использования представления. Шаблон “Visitor” Java

```
abstract public class Node  {
        public int lineno = 0;
        public int strPos = 0;
        abstract public void Accept(NodeVisitor visitor);
  }

public class ClassNode extends Node  {
        public String name = "class";
        public void Accept(NodeVisitor visitor) {
                visitor.visitClass(this);
                }
  }

public class AssignmentNode extends Node  {
        public String left = "left";
        public String right = "right";
        public void Accept(NodeVisitor visitor) {
                visitor.visitAssignment(this);
                }
  }
public class VariableNode extends Node {
        public String name = "var";
        public String initialValue = "right";
        public void Accept(NodeVisitor visitor) {
                visitor.visitVariable(this);
                }
```

```
public interface NodeVisitor  {
        public void visitClass(ClassNode node);
        public void visitAssignment(AssignmentNode node);
        public void visitVariable(VariableNode node);
}
```

Получаемая диаграмма классов примера



Спасибо за внимание!