Одиннадцатая независимая научно-практическая конференция «Разработка ПО 2015»

22 - 24 октября, Москва

Некоторые аспекты дизайна современного языка программирования общего назначения

Евгений Зуев, Университет Иннополис, Казань

Алексей Канатов, Исследовательский центр Samsung, Москва

Outline

- Введение
- Единица компиляции
- Композиция программ
- Альтернативный подход к наследованию
- Мультитипы
- Неинициализированные переменные и нулевые указатели
- Заключение

Новые языки: тенденции и потребности

Apple:
- 2014 **Swift** – язык общего назначения как замена Objective C

Facebook:
- 2014 **Hack** – замена PHP

Google:
- 2009 **Go** – язык реализации веб-приложений
- 2011 **Dart** – более надёжная и быстродействующая замена JavaScript

JetBrains (СПб, Россия):
- 2011 **Kotlin** – простая и эффективная замена Java

Microsoft:
- 2012 **TypeScript** - “лучший JavaScript” (аннотации типов, классы)

Mozilla:
- 2010 **Rust** – язык реализации для многоядерных архитектур

RedHat:
- 2011 **Ceylon** - “упрощённая Java”

Новые языки: тенденции и потребности

Apple:
- 2014 **Swift** – язык общего назначения как замена Objective C

Facebook:
- 2014 **Hack** – замена PHP

Google:
- 2009 **Go** – язык реализации веб-приложений
- 2011 **Dart** – более надёжная и быстродействующая замена JavaScript

JetBrains (СПб, Россия):
- 2011 **Kotlin** – простая и эффективная замена Java

Microsoft:
- 2012 **TypeScript** - “лучший JavaScript” (аннотации типов, классы)

Mozilla:
- 2010 **Rust** – язык реализации для многоядерных архитектур

RedHat:
- 2011 **Ceylon** - “упрощённая Java”

Почему?

- **Неудовлетворённость существующими ПО**
- **Несоответствие существующих ПО новым задачам и возросшим требованиям**
- **Желание сохранить контроль за эволюцией языка**
- **Маркетинговые и имиджевые резоны...**

Единицы компиляции

Три вида единиц компиляции

- **Анонимная процедура:** последовательность операторов
- **Standalone-подпрограмма**
- **Контейнер** (unit): поименованная совокупность атрибутов и подпрограмм
  - Может быть *параметризована* типами и/или константными выражениями
  - Для задания *типов*
  - Для конструирования *новых контейнеров* (наследование)
  - Для прямого *использования* атрибутов и подпрограмм в других контейнерах и standalone-подпрограммах.

Единицы компиляции

Три вида единиц компиляции

- **Анонимная процедура:** последовательность операторов
- **Standalone-подпрограмма**
- **Контейнер** (unit): поименованная совокупность атрибутов и подпрограмм
  - Может быть *параметризована* типами и/или константными выражениями
  - Для задания *типов*
  - Для конструирования *новых контейнеров* (наследование)
  - Для прямого *использования* атрибутов и подпрограмм в других контейнерах и standalone-подпрограммах.

```
StandardIO.put("Hello world!\n")

use StandardIO as io
routine(arguments: Array[String]) is
    io.put("Test!\n")
    c is C("This is a string“)
    io.put(c.string + "\n")
end

unit C
    string: String
    init (aString: like string) is
        string := aString
    end
end C
```

Единицы компиляции: вложенность

Общее правило

Любая единица может содержать единицы любого вида – в том числе того же вида

- Традиционная вложенность блоков (nesting)
- Локальные и вложенные контейнеры
- Вложенные функции

Максимальная простота и максимальная гибкость

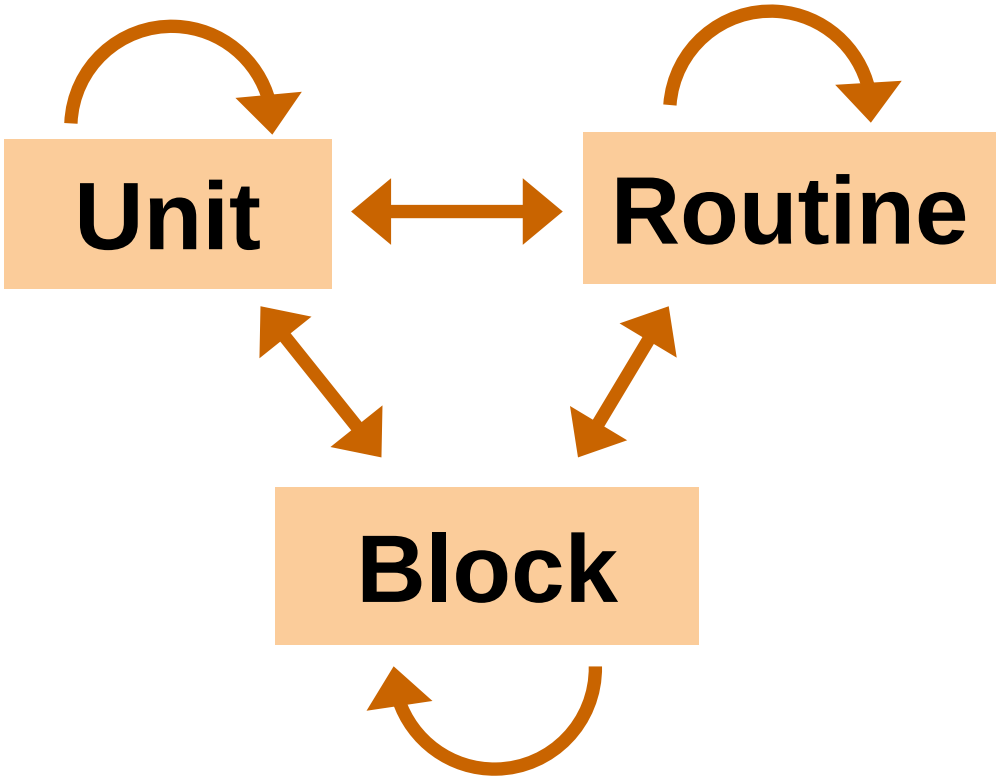
Единицы компиляции: вложенность

Общее правило

Любая единица может содержать единицы любого вида – в том числе того же вида

- Традиционная вложенность блоков (nesting)
- Локальные и вложенные контейнеры
- Вложенные функции

Максимальная простота и максимальная гибкость



Контейнер (unit)

Три способа применения контейнеров

Использование

Наследование

Типизация

Контейнер (unit)

Три способа применения контейнеров

Использование

Текущий контекст получает доступ к публичным свойствам A и B

Наследование

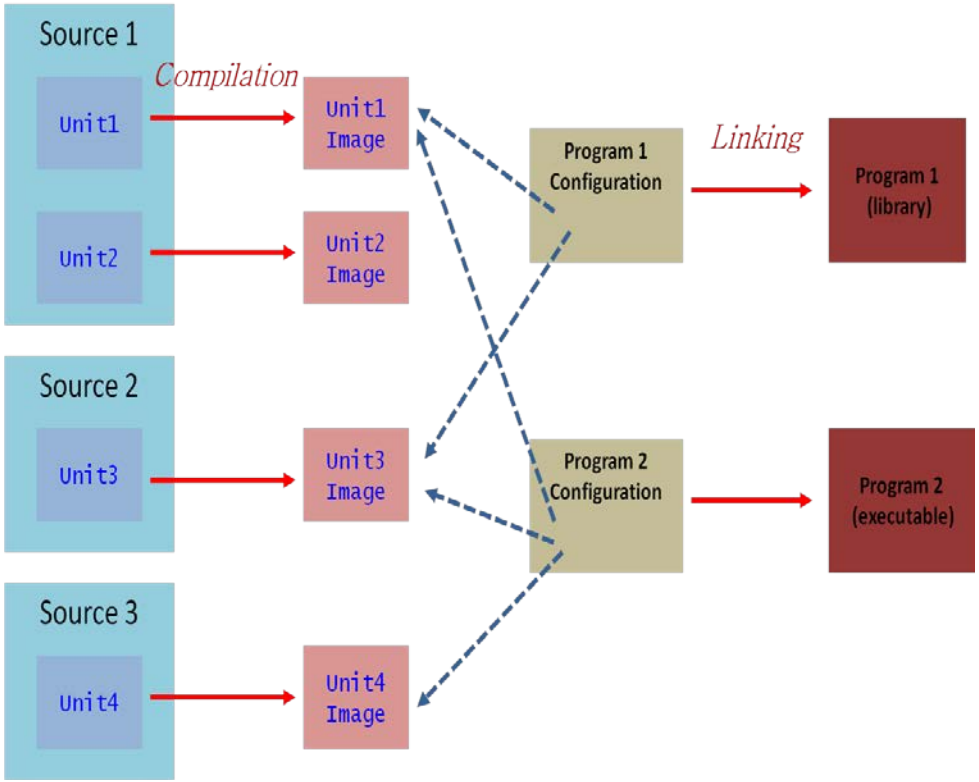
Контейнер C трактует контейнеры B и D как (базовые) классы, а контейнер E – как модуль

Типизация

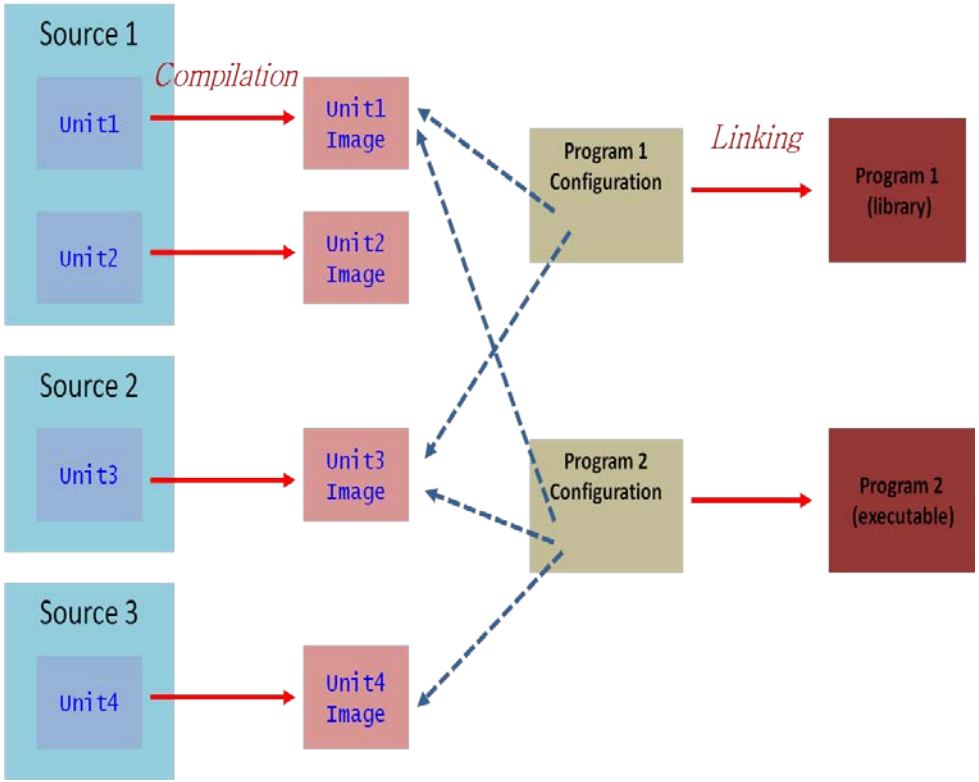
Атрибут x имеет тип A: контейнер A задает тип

```
use A, B as b
   // Использование юнита в качестве модуля

unit C extends B, ~D use E
      // Наследование
      // и эксклюзивное использование
      routine(argument:A) // A исп-ся как тип
      is
         E.foo()
      end
   end

routine is // standalone-процедура
   x : A
   C.routine(A) // C исп-ся как модуль
end

// Последовательность операторов –
// «анонимная подпрограмма»
routine()
```

Композиция программ



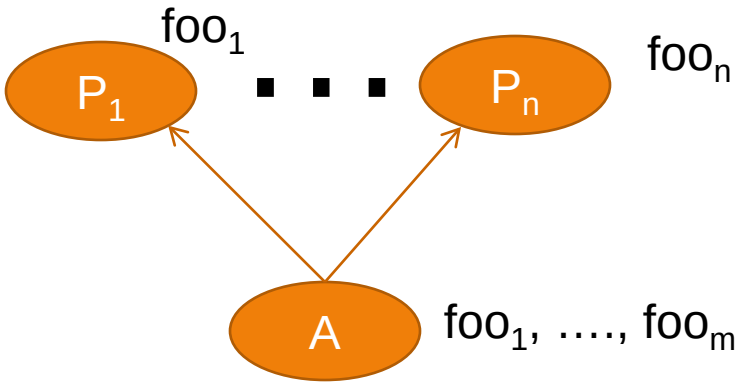
Композиция программ



- *Полное отделение* аспектов конфигурирования ПС и аспектов разработки ее компонентов.
- Семантика ЕК не должна зависеть от того, где находится используемая ею единица.
- Расположение исходного текста ЕК в каком-либо определенном источнике не должно влиять на ее отношения с другими единицами.
- Понятие «исходного текста» программы или ЕК не должно включаться в определение языка.
- Отсутствие «точки входа» в программу.
- Передача аргументов – через запросы к операционному окружению.

Альтернативный подход к наследованию

- Суть подхода: множественное наследование с перегрузками и конфликтами + проверка правильности обращения к свойствам.
- Необходимые проверки графа наследования при компиляции:
  - Отсутствие циклов в графе наследования
  - Разрешение конфликтов версий для полиморфизма
- Необходимые проверки при обращении к свойствам
  - Наличие свойств(а) и его доступность
  - Конформность типов параметров и аргументов
  - Cat calls




```
// P1, ..., Pn – родители A
a: A
a.foo(...)
  // Является ли данное
  // обращение правильным?
```

Мультитипы

Проблема:

Пусть имеются две или более сущности разных (не конформных друг другу) типов, с общими свойствами (features).

Как написать общий код для работы с этими свойствами, не вводя общего родителя (базового класса)?

Мультитипы

Проблема:

Пусть имеются две или более сущности разных (не конформных друг другу) типов, с общими свойствами (features).

Как написать общий код для работы с этими свойствами, не вводя общего родителя (базового класса)?

Решение:

Понятие **мультитипа**

Введение в язык этого понятия вместе с соответствующим механизмом контроля может заменить правила структурной эквивалентности типов без нарушения принципов статической типизации.

Мультитипы: пример

```
number: Integer | Real | myType
```

Атрибуту `number` можно присваивать сущности типов `Integer`, `Real`, `myType` или их наследников. Соответственно, можно обращаться к тем свойствам мультитипа, которые *присутствуют* во всех трех типах.

```
number := number1 + number2
```

Свойство сложения, которое обозначается инфиксной операцией `+`, должно присутствовать во всех типах, образующих мультитип.
Кроме того, вызов вида `number.+(number)` должен быть правильным для всех видов сочетаний `Integer.+(Integer)`, `Real.+(Real)` и `myType.+(myType)`.

Неинициализированные переменные и нулевые указатели

Нулевые (пустые) указатели (ссылки) – часть более общей проблемы: ошибки при попытке работы с *неинициализированными атрибутами*.

Три базовых принципа:

- Каждый атрибут должен получить значение до первого обращения к его свойствам.
- Если нужно описать атрибут без значения, то нельзя обращаться к его свойствам.
- Должен быть механизм безопасного перехода от неинициализированного атрибута к инициализированному.

Неинициализированные переменные и нулевые указатели

```
attr1 is 5       // Явная инициализация и неявная типизация
attr2: Integer   // Явная типизация и неявная инициализация
attr3: ?Integer // Явная типизация и отсутствие значения
attr1 := attr2; attr2 := attr1; attr3 := attr2 // Все валидно!
attr1 := attr3; attr2 := attr3 // Ошибка компиляции
if attr3 typeof Integer then
    // Внутри условного тип attr3 есть Integer.
    attr3 := attr3 + 5
    attr1 := attr3
end
?attr3 // Потеря значения и смена типа!
```

Заключение

В рамках выступления представлены

- Понятие контейнера (unit) и способы его использования
- Подход к композиции программ
- Альтернативный подход к наследованию
- Понятие мультитипа
- Решение проблемы корректности работы с атрибутами

В выступлении **НЕ** рассматривались ;-)

- Нотация (язык в целом)
- Поддержка функционального, параллельного и обобщённого (generic) программирования
- Backend & runtime (AOT vs JIT, GC vs RC, LLVM vs GCC, etc.)

СПАСИБО!

ВОПРОСЫ?