Разработка плагина для IntelliJ IDEA, добавляющего поддержку языка Frege

Карнаухов К. Е.,
Худяков Ю.А.
Сурков П.А.

Ментор: Атамась Семён Михайлович

18 июня 2021 г.

Коротко о Frege

```
module Hello where
greeting friend = "Hello, " ++ friend ++ "!"
main args = do
    println (greeting "World")
```

Frege и Java

```
data JArrayList = native java.util.ArrayList where
    native add :: Mutable s (JArrayList e) -> Int -> e -> ST s ()
               | Mutable s (JArrayList e) -> e -> ST s Bool
```

Где писать на Frege?

- Текстовые редакторы (только подсветка синтаксиса)
- Плагин для Eclipse (странный функционал)
- Плагин для IDEA 2016 года (30 коммитов)

JB Plugin Ideas

Most Requested

Frege Programming language

Dierk Koenig

Frege is a programming language that brings purely functional programming like Haskell to the JVM with some Java-integration specifics: http://www.frege-lang.org. As indicated by its

259 102 04.10.2014

Задачи

- Сурков Пётр
  - Лексер (в т.ч. layout rule)
  - Парсер
  - Автодополнение ключевых слов и live templates
  - Поддержка документации
- Худяков Юрий
  - Система сборки и компиляции
  - Интерпретатор
  - Система типов
- Карнаухов Кирилл
  - Навигация
  - Индексация файлов
  - Автодополнение на основе навигации
  - Подсветка кода и настройка цветов
  - Немного Quick Fix и переименование

Lexer: JFlex

```
digit                   = [0-9]
digitNonZero            = [1-9]
signs                   = [+-]
number                  = {signs}?(0 | {digitNonZero}{digit}*)

{number}                    { return FregeTypes.NUMBER; }
```

Layout rule

- Можно писать так:

```
f = do {let {apples = 32; bananas = 10}; println apples + bananas}
```

- Но обычно пишут так:

```
f = do
    let apples = 29
        bananas = 13
    println $ show (apples + bananas)
```

- Эти стили можно комбинировать
- Всего есть 7 правил, задающих layout rule

Layout rule: решение

- Сделаем следующую биекцию:

  { = OPEN
  ; = END DECL
  } = END SECTION

- Обёртка над лексером, расставляющая вирт. токены
- Получим следующий код:

```
f = do OPEN
    let OPEN  apples = 29 END DECL
              bananas = 13 END SECTION END DECL
    println $ show (apples + bananas) END SECTION
```

Lazy layout lexer

- Иногда вставлять виртуальные токены надо в часть, которая уже была обработана лексером
- В IDEA не предусмотрено такое поведение
- Инкрементальный и ленивый лексинг невозможен
- Программу можно специальным образом бить на блоки и их уже лексить лениво

Парсер: Grammar-kit

```
newtypeDecl ::= ABSTRACT? NEWTYPE conidUsage typedVarid* EQUAL construct whereSection? {
    pin=2
    implements=["com.plugin.frege.psi.FregePsiClass"
                "com.plugin.frege.psi.FregeTypeParametersHolder"]
    mixin="com.plugin.frege.psi.mixin.FregeNewtypeDeclMixin"
    stubClass="com.plugin.frege.stubs.FregeClassStub"
}
```

Тестирование и результаты

- 3 вида тестов:
  - Файл должен распарситься
  - Файл не должен распарситься
  - Построенное PSI дерево совпадает с образцом
- Больше 70 тестов
- Парсится вся стандартная библиотека, и все инструменты по работе с Frege, за исключением одного файла под старую версию компилятора

Keyword completion & live templates

- Автодополнение ключевых слов и написание шаблонов на основе построенного дерева

Наша система сборки

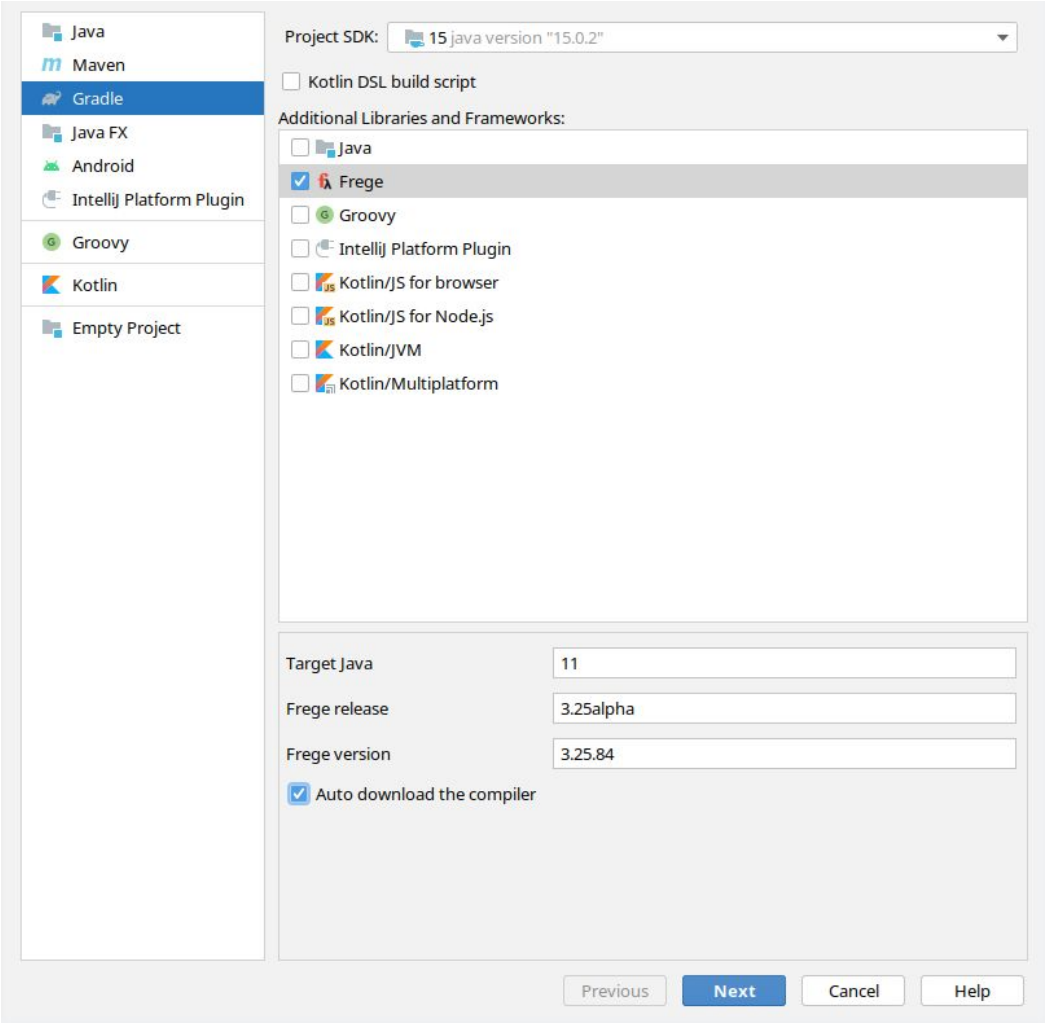
- Основана на Gradle
- Имеет хорошую совместимость с другими JVM-языками и зависимостями из них
  - По модулю того, что сам язык не поддерживает сложные мультиязычные проекты
- Остается гибкой, легко изменяется пользователем

Как устроена наша система сборки?

- Создание нового проекта с указанием настроек
  - Шаблонный build.gradle файл
  - Настройки в gradle.properties
- Автоматическое скачивание компилятора

- Сборка других языков
- Сборка Frege
- Запуск на Java Virtual Machine

Создание нового проекта



Интерпретатор

Интерпретатор (или REPL = Read-Eval-Print-Loop)

build.gradle | Main.fr

```
1  module pkg.Main where
2
3  myVariable = 1
4
```

Run: REPL

```
/usr/lib/jvm/jdk-15.0.2/bin/java -Dfile.encoding=UTF-8 -jar /home/yourass/Desktop/proga/hse19/projects/intellij-frege/othe
Welcome to Frege 3.25.84 (Oracle Corporation Java HotSpot(TM) 64-Bit Server VM, 15.0.2)

frege> :load /home/yourass/Desktop/proga/hse19/projects/intellij-frege/other/GradleBased96/src/main/frege/pkg/Main.fr
Module pkg.Main

frege> myVariable
1

frege> f x = [x, x + myVariable]
function f :: Int -> [Int]


frege> myList = [1, 3, 5] >>= f
```

Контекст интерпретатора

Name: REPL

☐ Allow parallel run ☐ Store as project file

Root module: other.GradleBased96

REPL Arguments:

Loaded modules

Modules to Load
☑ MySeparateModule1
☐ MySeparateModule1.test
☑ MySeparateModule1.main
☑ MySeparateModule2
☐ MySeparateModule2.test

Запуск кода из произвольного места в проекте

```
5   myVariable2 = 2
6
7
```

Choose REPL Action

Run in existing REPL

Create new REPL and run text in it

Система типов

- Сложно реализовывать самим
- Можно брать из компилятора
  - Сложный код, не смогли разобраться с некоторыми багами
- Можно отдавать интерпретатору, он умеет выводить тип
  - Это ещё не успели до конца реализовать

$$\boxed{Q_0 ; \Gamma \mapsto M}$$

$$\frac{}{Q_0 ; \Gamma \mapsto \epsilon}\ \text{EMPTY}$$

$$\frac{\begin{array}{c}\Gamma \mapsto e : \tau \rightsquigarrow C \quad Q_0 ; \epsilon ; \epsilon \vdash solve(C) \rightsquigarrow Q ; \theta \\ \overline{a}\ \text{fresh} \quad \overline{\alpha} = fuv(\theta\tau, Q) \\ Q_0 ; \Gamma, (x{:}\forall\overline{a}.[\overline{\alpha \mapsto a}](Q \Rightarrow \theta\tau)) \mapsto M\end{array}}{Q_0 ; \Gamma \mapsto \texttt{let}\ x = e, M}\ \text{BIND}$$

$$\frac{\begin{array}{c}\Gamma \mapsto e : v \rightsquigarrow C \\ Q_0 ; Q ; \epsilon \vdash solve(C \wedge v \sim \tau) \rightsquigarrow \epsilon ; \theta \\ Q_0 ; \Gamma, (x{:}\forall\overline{a}.Q \Rightarrow \tau) \mapsto M\end{array}}{Q_0 ; \Gamma \mapsto \texttt{let}\ x :: (\forall\overline{a}.Q \Rightarrow \tau) = e, M}\ \text{ABIND}$$

$$\boxed{Q_0 ; Q_{given} ; \overline{\tau}_{untch} \vdash solve(C_{wanted}) \rightsquigarrow Q_{residual} ; \theta}$$

$$\frac{\begin{array}{c}Q_0 ; Q_g ; \overline{\tau} \vdash simpleS(\mathbf{simp}[C]) \rightsquigarrow Q_r ; \theta \\ \forall([\overline{\tau}_i](Q_i \supset C_i) \in \mathbf{prop}[\theta C]), \\ Q_0 ; Q_g Q_r Q_i ; \overline{\tau}\overline{\tau}_i \vdash solve(C_i) \rightsquigarrow \epsilon ; \theta_i\end{array}}{Q_0 ; Q_g ; \overline{\tau} \vdash solve(C) \rightsquigarrow Q_r ; \theta}\ \text{SOLVE}$$

$$\boxed{Q_0 ; Q_{given} ; \overline{\tau}_{untch} \vdash simpleS(Q_{wanted}) \rightsquigarrow Q_{residual} ; \theta}$$

... parameter X solver ...

Figure 4: Top-level algorithmic rules

```
frege> :t func
(Applicative a,Bind a) => a Int -> a [Int]
```

Навигация

- Навигация основана на ссылках к определению
- Задача: найти, куда указывают ссылки

```
sumOfTwo   a b   = a + b
sumOfThree a b c = sumOfTwo a $ sumOfTwo b c
```

Навигация: простой пример

Иногда разрешить ссылку не так сложно:

Test.fr

```
1  module Test where
2
3  import frege.data.HashMap
4
5  main = undefined
6
```

Навигация: переменная (1/2)

```
getValue a b c = calculateMeanValue (a + b) (a - c)
    where calculateMeanValue x y = x - y + a
```

```
calculate a b c = (getMean a b c) + processSum (a + b + c)
    where processSum n = case n of
                0 → -1
                m → tryProcess m
                    where tryProcess l = getMean l l l
          getMean x y z = (x + y + z) / 3
```

Навигация: переменная (2/2)

Test.fr × Other.fr ×

```
1  module examples.Test where
2
3  import another.Other as O (importantFunction)
4
5  main args = println $ O.importantFunction 10
6
7
8
```

Навигация: квалификатор

```
test = MyClass.func 42

class MyClass a where
    func :: a → a → a
    func _ = undefined

instance MyClass Int where
    func x y = x + y
```

```
test = Int.func 42

class MyClass a where
    func :: a → a → a
    func _ = undefined

instance MyClass Int where
    func x y = x + y
```

Навигация: в Java (и другие JVM-языки)

Test.fr

```
1   module examples.Test where
2
3   data JArrayList = native java.util.ArrayList where
4       native add :: Mutable s (JArrayList e) → Int → e → ST s ()
5                  | Mutable s (JArrayList e) → e → ST s Bool
6
7       native addAll :: Mutable s (JArrayList e) → Int → Mutable s (JCollection e) → ST s Bool
8
9   data JCollection = native java.util.Collection -- TODO
10
```

Навигация: из Java (и других JVM-языков)

JavaTest.java × Other.fr ×

```
1   package test;
2
3   import another.Other;
4
5   public class JavaTest {
6       public static void main(String[] args) {
7           Other.importantFunction( arg0: 10);
8       }
9   }
10
```

Навигация: суммарно

- Реализована навигация для подавляющей части конструкций языка
- Более 90 тестов
- Разрешается около 92% из 40000 поддерживаемых ссылок в стандартной библиотеке

Индексация

- Stub – сжатый бинарный формат для элементов из дерева
- Хранит в себе основную информацию об элементе
- Позволяет не парсить одну и ту же часть кода несколько раз
- Позволяет навигироваться намного быстрее

Переименование переменных

- Элементы с ссылками умеют переименовываться
- Для этого также написаны тесты

Other.fr | Mine.fr

```
module another.Other where

getFirst x y   = x
getSecond x y  = getFirst y x
getThird x y z = getFirst z x
```

JavaTest.java

```
package test;

import another.Other;

public class JavaTest {
    public static void main(String[] args) {
        var result :Object = Other.getFirst(1, 2);
    }
}
```

Quickfix

Other.fr × Mine.fr ×

```
1  module another.Other where
2
3  test = justMine
4
```

Автодополнение (1/3)

```
function1 = 1
function2 = 2
function3 = 3

test = functi
```

- function1
- function2
- function3

Press Ctrl+. to choose the s

Автодополнение (2/3)

```
other :: Int → Int → In
```

- Int
- IndexOutOfBoundsException
- InputMismatchException
- InputStream
- InputStreamReader
- InstantiationException
- Integer
- Integral
- InterruptedException
- InvalidPropertiesFormatException
- ExceptionInInitializerError

Press Enter to insert, Tab to replace Next Tip

Автодополнение (3/3)

Other.fr

```
1  module another.Other wh Paused...
2
3  import mine.Mine
4
5  test = justMin
6
7
8
```

justMine

Press Enter to insert, Tab to replace Next Tip

Mine.fr

```
1  module mine.Mine where
2
3  justMine x = 3 + x
4
```

Поиск использований (find usages)

```
data Choice a b = First a | Second b

getFirst :: Choice I
getFirst (First x)
getFirst (Second _)

getSecond :: Choice Int → Int
getSecond (Second x) = x
getSecond (First _)  = error "cannot choose left"
```

Usages of First() — Results in 'A

*	Other.fr	3	data Choice a b = **First** a | Second b
	Other.fr	6	getFirst (**First** x) = x
	Other.fr	11	getSecond (**First** _) = error "cannot choose left"

Подсветка кода

- Подсветка в двух основных темах: темная и светлая
- Возможность кастомизации цветов
- Подсветка не только на основе лексера, но также и на основе конкретных синтаксических структур

Подсветка кода: настройка цветов

Scheme: **IntelliJ Light**

- Bad value
- Block comment
- Brackets
- Char
- Function name
- Keyword
- Line comment
- Number
- Operator
- String
- Type
- Type parameter
- Undefined

☑ Bold ☐ Italic

☑ Foreground 00627A

☐ Background

☐ Error stripe mark

☐ Effects

Bordered

☐ Inherit values from:

Classes→Instance method
(Language Defaults)

```
-- https://github.com/Frege/frege/blob/master/examples/Grep.fr--- A simple grep
module examples.Grep where

--- exception thrown when an invalid regular expression is compiled
data PatternSyntax = native java.util.regex.PatternSyntaxException
derive Exceptional PatternSyntax
```

Подсветка кода: темная тема

```
--- A simple grep
module examples.Grep where

--- exception thrown when an invalid regular expression is compiled
data PatternSyntax = native java.util.regex.PatternSyntaxException
derive Exceptional PatternSyntax

main [] = stderr.println "Usage: java examples.Grep regex [files ...]"
main (pat:xs) = do
        rgx ← return (regforce pat)
        case xs of
            [] → grepit rgx stdin
            fs → mapM_ (run rgx) fs
     `catch` badpat where -- it is just a comment!
        badpat :: PatternSyntax → IO ()
        badpat pse = do
            stderr.println "The regex is not valid."
            stderr.println pse.getMessage

myTest = undefined
```

Подсветка кода: светлая тема

```
--- A simple grep
module examples.Grep where

--- exception thrown when an invalid regular expression is compiled
data PatternSyntax = native java.util.regex.PatternSyntaxException
derive Exceptional PatternSyntax

main [] = stderr.println "Usage: java examples.Grep regex [files ...]"
main (pat:xs) = do
        rgx ← return (regforce pat)
        case xs of
            [] → grepit rgx stdin
            fs → mapM_ (run rgx) fs
    `catch` badpat where -- it is just a comment!
        badpat :: PatternSyntax → IO ()
        badpat pse = do
            stderr.println "The regex is not valid."
            stderr.println pse.getMessage

myTest = undefined
```

Поддержка документации

- Поддержка документации на основе ссылок

```
f :: Int
f = 0
```

Ближайшие планы

- Увеличивать количество разрешающихся ссылок в стандартной библиотеке
- Работать с системой типов
- Улучшить автодополнение
- Разместить плагин в JB Marketplace

Ссылки

- Наш репозиторий:
  github.com/IntelliJ-Frege/intellij-frege
- Репозиторий Frege:
  github.com/Frege/frege
- Спецификация Frege (устаревшая, 2014 года):
  frege-lang.org/doc/Language.pdf
- JB Plugin Ideas:
  plugins.jetbrains.com/plugin-ideas
- JFlex:
  jflex.de
- Grammar-Kit:
  github.com/JetBrains/Grammar-Kit

Lexer: многозначность точек

- Точка как оператор композиции `f . g`
- Точки в range списков `[1..10]`
- Точками можно заканчивать float `x = 1.`
- Точками задаются полные имена `Data.TreeMap`
- Точками задаются primary выражения `people.[42].age`
- Операторы с точкой в начале `.|.`

Система сборки

Система сборки:

- Позволяет собрать и запустить проект
- Поддержка зависимостей

Основные системы сборки JVM языков:

- Gradle – современная
- Maven – старая, но используется
- Ant – совсем старая, почти не используется
- Sbt – только Scala

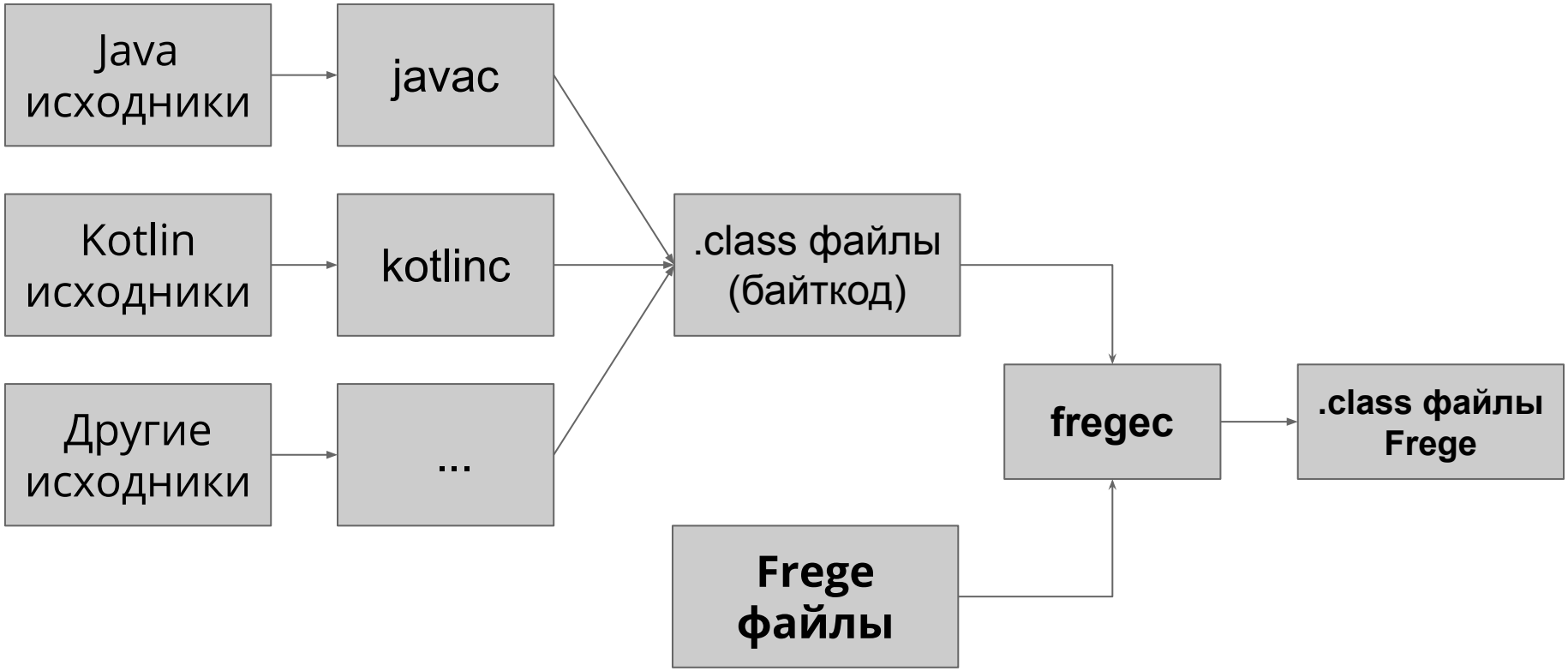
Аналоги системы сборки

- Gradle плагин от создателей языка – старый и редко обновляется, не собирается
- Maven плагин – старый и не обновляется, не собирается
- Система сборки старого плагина для IDEA – написана на Java, поддерживает сборку только Frege файлов, не работает, так как изменился компилятор

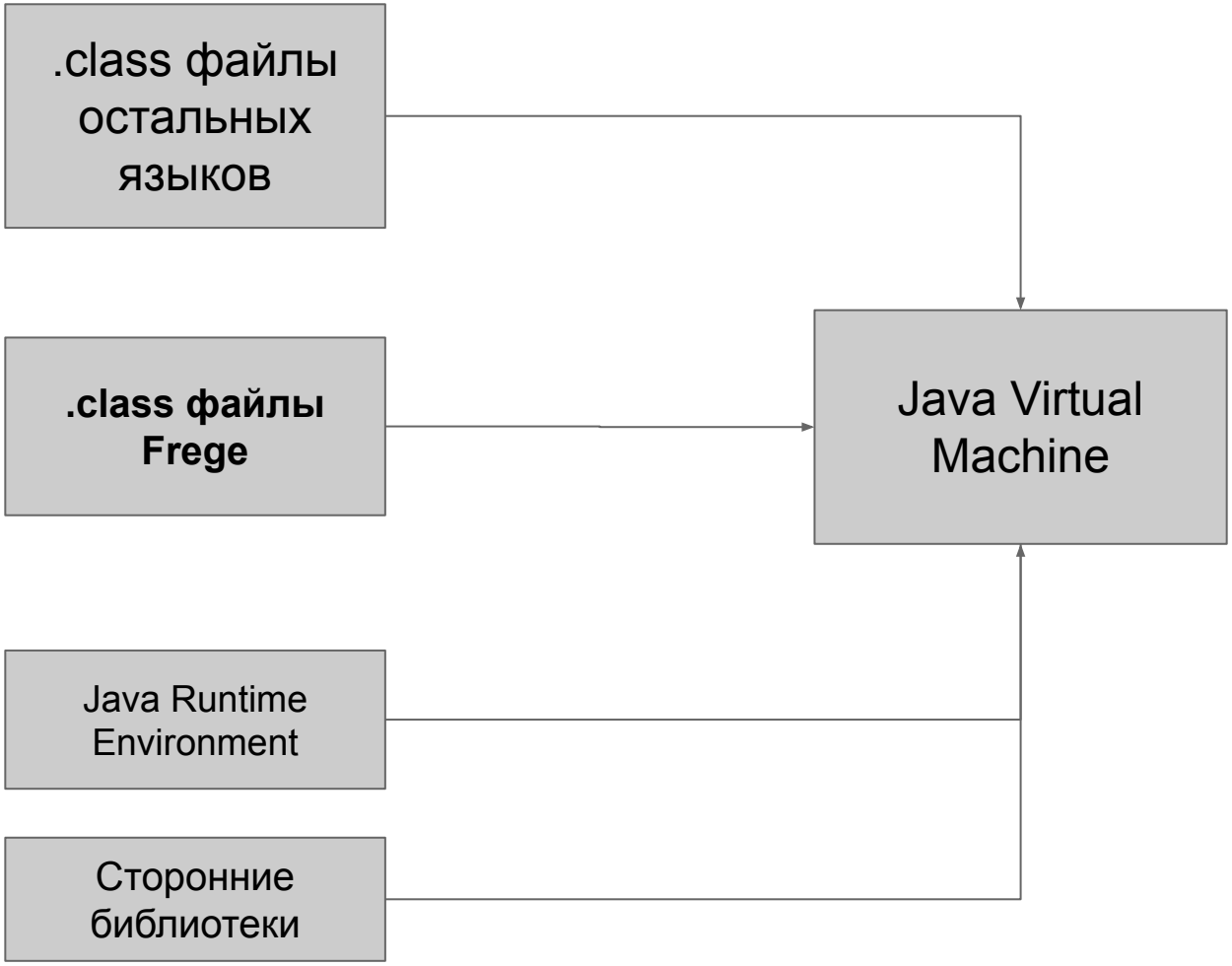
Как устроена наша система сборки?

- Запуск компилятора Java для Java-зависимостей
  - Получаем .class файлы с JVM-байткодом
- Запуск компилятора Frege
  - Использует .class файлы Java
  - Создает свои .class файлы
- Запуск скомпилированных .class файлов с помощью JVM

Как устроена наша система сборки?



Как устроена наша система сборки?

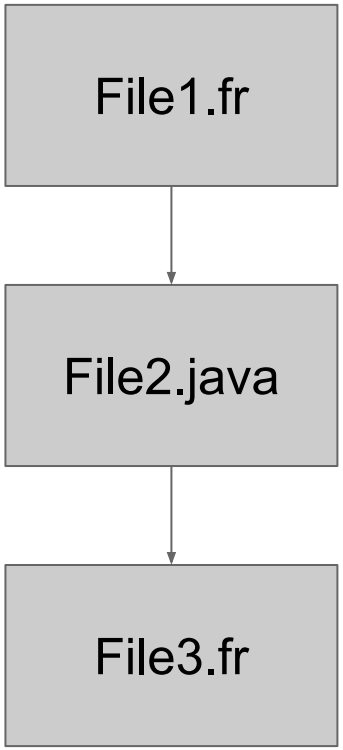


Проблемы сборки

Граф без циклов

Можно собрать вручную в правильном порядке

Компилятор Frege не умеет

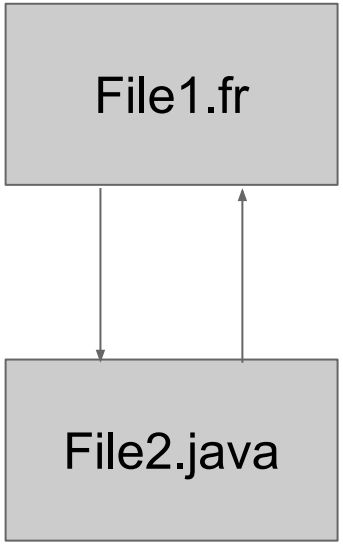


Пример

Проблемы сборки

Граф с циклами

Корректно собрать нельзя



Пример

Интерпретатор

- Подсветка кода, автодополнение, навигация и т.д.
- Контекст проекта
- Запуск кода из произвольного места в проекте

Индексация: мотивация

Мы смотрели на разрешение ссылки у импорта. А как получить элемент из другого файла?

- Самая очевидная идея: обойти все файлы проекта (включая стандартную библиотеку) и там поискать
- Слишком медленно – ждать несколько секунд в лучшем случае
- Почему, если поменялась другая часть проекта, то при разрешении той же самой ссылки придется снова ждать несколько секунд? Неудобно

Индексация

- В IntelliJ-IDEA для такой цели есть так называемые stub’ы
- Они хранят основную информацию об элементе (например, имя)
- Имеет смысл сохранять только ту информацию, которая доступна из других файлов
- В нашем плагине сохраняются имена всех методов, а также полные и сокращенные имена всех классов
- Также сохраняется список импортов для каждого модуля (специфика языка)