

АЛЬТЕРНАТИВНЫЕ ЯЗЫКИ ДЛЯ JVM

Лекция 1

ПРЕДМЕТ ИЗУЧЕНИЯ

- Scala - out of scope
- Kotlin - примерно 7-8 занятий
- Groovy - 2-3 занятия
- Clojure - 2-3 занятия

МОИ ПРЕДПОЛОЖЕНИЯ ОБ АУДИТОРИИ

- Хардкорные разработчики
- Пишут код давно и увлеченно, пробовали не один язык
- Знают базовые концепции вроде ООП и ФП
- Важно, что там "под капотом"
- Знают Java
- Очень возможно, что уже что-то писали на Kotlin

ЧТО ПРО KOTLIN

- Не ликбез, не обучение программированию
- 3 лекции на базовый язык
- 3 лекции на корутины
- Что-то еще, варианты: ФП на Kotlin, аннотации-интроспекции-рефлексии, DSL, веб, плагины, другое ?
- Disclaimer: я не мобильный разработчик

ПРАВИЛА КУРСА

- "Внутренняя" оценка - 100 балльная шкала
- Для ITMO будут уточняться правила перевода баллов
- Для вышки - нелинейный переход !!!
- Не делением на 10 !!!

ПРАВИЛА КУРСА

- $(50, 57]$ - 4 в 10-бальной
- $(57, 64]$ - 5 в 10-бальной
- $(64, 71]$ - 6 в 10-бальной

ПРАВИЛА КУРСА

- $(71, 78]$ - 7 в 10-бальной
- $(78, 85]$ - 8 в 10-бальной
- $(85, 92]$ - 9 в 10-бальной
- $(92, \text{Inf})$ - 10 в 10-бальной

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- Домашние задания с ревью (но автоматизированные проверки возможны)
- Автоматизированные короткие задания (stepik и т.п.) (выборочное ревью возможно)
- "Экзамен" - скорее собеседование (5 экспресс-вопросов)
- Бонусы за активность и особо красивые решения
- Штрафы за списывание и нарушение правил

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- 4-5 домашек
- Каждая оценивается на 100 баллов
- На балл влияют: полнота решений, качество (по ревью), соблюдение дедлайнов
- Считаем среднюю по домашкам, отбросив худшую
- В итоговую оценку идет с коэффициентом 0.5

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- После 1-2 семинаров - набор автоматизированных тестов
- Каждый набор тестов оценивается в 100 баллов
- Считаем среднюю, отбросив 2 худших
- В итоговую оценку идет с коэффициентом 0.2

СОСТАВНЫЕ ЧАСТИ ОЦЕНКИ

- Экзамен - как беседа
- Что-то спрошу по домашкам, попрошу кусочек кода поревьюить
- Что-то спрошу на понимание
- Пять вопросов, по 20 баллов каждый
- В итоговую оценку идет с коэффициентом 0.3

МОЖНО БЕЗ ЭКЗАМЕНА

- Первый способ - получить высокие баллы за домашки и тесты
- Тогда 0 за экзамен не мешает зачету
- Но балл будет 4-5, ну максимум 6
- Второй способ - получить право на автомат
- Выдается за высокую (и разумную) активность на семинаре
- На фоне высоких показателей по домашкам и тестам

ПОДРОБНЕЕ ПРО ТЕСТЫ

- Короткий строгий дедлайн
- Разбор особо интересных задач на следующем семинаре
- В общем случае запоздалые доделки не предусмотрены
- Если пропускается по уважительной причине, то возможно новое задание
- Новое задание выдается в конце семестра

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Дедлайн никто не меряет на секунды
- По мере пропущенных дней набегает понижающий коэффициент
- 1 день - 0.95
- 2 дня - 0.9
- 14 дней 0.3
- Понижается в день на 0.05

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Через 14 дней фиксируется
- Остается 0.3 навсегда
- В случае ухода на пересдачу - не восстанавливается !!!
- Мораль: не забрасывайте домашки !!!
- Есть 5 дней на индивидуальное продление дедлайна

ПОДРОБНЕЕ ПРО ДОМАШКИ

- К заданиям могут быть тесты - их надо проходить
- Тесты могут быть закрытыми
- Задание, на котором не проходят тесты - не ревьюится и оценивается в 0
- Списывать - нельзя
- Делиться своими тестами - можно абсолютно легально

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Если тесты пройдены или не предусмотрены, то начинается code review
- По итогам review балл может быть понижен
- Как правило, замечания можно исправлять
- Как правило, со скидкой
- Как правило, одна итерация на исправление

ПОДРОБНЕЕ ПРО ДОМАШКИ

- Возможны бонусы
- За активность на семинаре, доделки заданий с семинаров (будет объявляться)
- За особо красивые решения в тестах
- Бонусные баллы добавляются к релевантным домашкам, но не выше 100 баллов
- Переноса на другие домашки нет

АНТИПАТТЕРНЫ

- Появиться внезапно в конце семестра в расчете на особые условия
- Заявить задним числом, что "ничего не было понятно"
- Для тех, кто переводится - воспринимать перевод как универсальную индульгенцию
- Апеллировать к правилам других курсов
- Молча исчезнуть в середине семестра - лучше дать обратную связь

ПРАВИЛЬНЫЕ ПАТТЕРНЫ

- Вовлеченно участвовать в семинарах
- Задавать вопросы
- Давать конструктивную обратную связь

МОИ ОБЯЗАТЕЛЬСТВА

- Прислушиваться к обратной связи - в рамках реального
- Отвечать на вопросы и обратную связь - в рамках разумного
- Давать намеки и разумные подсказки по мере необходимости
- Возможно, незначительно корректировать критерии в сторону смягчения

К ДЕЛУ

- Java - язык, Java VM - платформа для исполнения
- Открытая спецификация. Не обязательно получать байткод из Java
- Были проекты Jython, JRuby, не сложилось
- Scala - для любителей статического ФП

К ДЕЛУ

- Clojure - для любителей динамического ФП
- Groovy - своего рода питончик
- Kotlin - понемножку от разных парадигм
- С упором на удобство программирования

ПЛАН НА СЕГОДНЯ

- Первое погружение в языковую среду
- Захватим несколько концепций
- Потом по ним подробнее пройдемся
- Задача: писать небольшие куски кода и их понимать

БАЗОВАЯ СТРУКТУРА KOTLIN-ПРОГРАММЫ

- В JVM не бывает кода вне класса
- А в Kotlin - бывает
- Компилятор оборачивает внеклассвые функции в служебные классы
- Простейшая программа: `fun main() {}`

ГДЕ И КАК ПИСАТЬ

- Типовой вариант - IDEA
- Через maven/gradle
- Но есть CLI-утилиты (не сходу видны)
- Можно компилировать из командной строки
- REPL, скрипты - тоже есть

БАЗОВАЯ СТРУКТУРА ПРОГРАММЫ

- Компилятор породит класс по имени файла
- `main.kt` породит класс `MainKt`
- В нем будет публичный статический метод `main()`
- И для `main` без аргументов будет порожден JVM-метод `main(String[] args)`
- Тоже публичный

HELLO, WORLD

- Хотим что-то напечатать
- На уровне JVM все через класс
- Формально канонический вариант печати:
`System.out.println("Hello")`
- Но есть персональный сахар на такой случай
- `fun main() { println("Hello, world"); }`

УСЛОЖНИМ ЧУТЬ-ЧУТЬ

```
1 fun main(args: Array<String>) {  
2     println("Hello: " + args[0])  
3 }
```

ПРОАНАЛИЗИРУЕМ

- Точки с запятой можем убирать, если переносится строка
- Появляется новый тип данных
- Похоже на коллекцию с дженериками
- В IDEA можем даже попасть в класс Array

ПРОАНАЛИЗИРУЕМ

- С другой стороны - в байткоде увидим, что это JVM-массив
- Некоторые классы ведут двойную жизнь
- В контексте исходного кода это обычные классы
- А во время исполнения - это особые случаи из JVM

ОСОБЫЕ КЛАССЫ

- Помимо массивов это примитивные JVM-типы
- Int, Long, Byte, Char, Float, Double, Boolean
- В исходниках это классы
- Не те, которые наследуют `java.lang.Number`
- Во время исполнения - примитивные типы (но есть нюансы)

ИНТЕРЕСНАЯ КОЛЛИЗИЯ

- Все стандартные Java-классы доступны в Kotlin
- Классы из `java.lang` - доступны без `import`-тов
- И есть свои импорты по умолчанию
- То есть есть `Int` (как Kotlin понимает целое число) и есть `Integer` (это `java.lang.Integer`)
- `Boolean`, `Short` и `Long` - короткие имена совпадают (приоритет, конечно же, за Kotlin-типом)

ОБРАЩЕНИЕ ПО ИНДЕКСУ

- Индекс в квадратных скобках намекает на JVM-массив
- Но это не так существенно
- К коллекциям тоже обращаемся через квадратные скобки
- И для Array можно использовать `get`
- Байткод не поменяется

СТРОКОВАЯ ИНТЕРПОЛЯЦИЯ

- IDEA жалуется на аргумент `println`
- Для порождения параметризованных строк есть механизм интерполяции
- `println("Hello: ${args[0]}")`
- Давайте что-нибудь посчитаем

КОЛИЧЕСТВО ГЛАСНЫХ В СЛОВЕ

УСЛОЖНИМ ЧУТЬ-ЧУТЬ

```
1 fun main(args: Array<String>) {  
2     val s = args[0]  
3     var i = 0  
4     var count = 0  
5     while (i < s.length) {  
6         val c = s[i]  
7         if (c == 'a' || c == 'e' || c == 'i' ||  
8             c == 'o' || c == 'u' || c == 'y') {  
9             count += 1  
10        }  
11    }  
12    println(count)  
13 }
```

РАЗБЕРЕМСЯ В КОДЕ

- (Это не самый идиоматичный код)
- Считаем за отправную точку
- Мы видим объявления переменных, но не видим типов
- Это нормально и идиоматично
- Тип определяется компилятором
- Kotlin - статический язык со строгой типизацией

РАЗБЕРЕМСЯ В КОДЕ

- Видим два варианта объявления: `var` и `val`
- То, что объявлено как `var`, может меняться
- В Java-строке нет поля `length`, но есть метод `length`
- А вот в Java-массиве есть поле `length`
- А здесь как будто такое поле появляется у строки

PROPERTY

- Property - элемент объекта класса
- Вычисляется на основе состояния объекта
- Обозначается как имя без скобок
- И есть синтаксис для их определения
- А здесь как будто такое поле появляется у строки

PROPERTY

- Но String на особом положении
- Компилятор его обрабатывает особо
- И он непосредственно подставляет вызов JVM-метода `length()` на место `property length`
- А в обычных `property` мы в Kotlin будем определять `get-er`

ОБРАЩЕНИЕ К СИМВОЛУ В СТРОКЕ

- Обращаемся по индексу
- Как к элементу массива
- Можно вызвать `get` - как для `Array`
- А вот `charAt` - как в `Java` - нельзя

ОБЩАЯ ИДЕЯ

- А в байткоде увидим вызов `charAt`
- Общая идея: унификация Java-наследия
- Местами используются ситуативные подпорки
- Но есть и возможности языка, которые можно переиспользовать в своих ситуациях
- И их можно использовать самостоятельно - окультурировать старые Java-библиотеки

ОБЩАЯ ИДЕЯ

- Пример универсальной возможности: определение новых методов над существующими классами
- С ограничениями: никто нас не пустит к приватным элементам чужого класса
- Пример ситуативной подпорки: замена `get` на `charAt` в строке
- Без нее был бы дополнительный метод в синтетическом классе

УЛУЧШИМ КОД

```
1 fun main(args: Array<String>) {  
2     val s = args[0]  
3     var count = 0  
4  
5     s.forEach { c ->  
6         if (c == 'a' || c == 'e' || c == 'i' ||  
7             c == 'o' || c == 'u' || c == 'y') {  
8             count += 1  
9         }  
10    }  
11  
12    println(count)  
13 }
```

ОСМЫСЛИМ

- Похоже на lambda-style в Java
- Только в Java нет потоков над строками
- И в Kotlin тоже нет
- В байткоде это итерация по CharSequence
- Но чем меньше var в коде, тем лучше

ОСМЫСЛИМ

- И еще нет скобок после `forEach`
- Очень полезный сахар
- Если единственный параметр - `lambda`, то скобки опускаются
- Если есть другие параметры перед ней, то скобка закрывается после предпоследнего
- Способствует DSL-стилю

УЛУЧШИМ КОД

```
1 fun main(args: Array<String>) {  
2     val s = args[0]  
3     val count = s.filter { c ->  
4         c == 'a' || c == 'e' ||  
5         c == 'i' || c == 'o' ||  
6         c == 'u' || c == 'y'  
7     }.length  
8     println(count)  
9 }
```

ОСМЫСЛИМ

- Выпилили `var`
- В реализации увидим, что материализуется строка
- На больших данных может быть проблемой
- Решение узнаем позже
- Можно сравнение сделать покомпактнее

УЛУЧШИМ КОД

```
1 fun main(args: Array<String>) {  
2     val s = args[0]  
3     val VOWELS = "aeiouy".toSet()  
4     val count = s.filter {c -> c in VOWELS}.length  
5     println(count)  
6 }
```

ОСМЫСЛИМ

- Совсем красиво, но есть нюанс
- Set - это HashSet из Java над `java.lang.Character`
- Не идеально по скорости
- Дорого по памяти, если таких множеств много
- Но для многих ситуаций - ОК
- Компетенция хорошего разработчика - отличать эти варианты

ПОРАБОТАЕМ С ЧИСЛАМИ И С ФУНКЦИЯМИ

- Передаем число через командную строку
- Напечатаем разложение на простые множители
- По множителю на строку
- Вынесем реализацию в функцию

НАЧАЛО

```
1 fun main(args: Array<String>) {  
2     printProduct(args[0].toLong())  
3 }  
4  
5 fun printProduct(n: Long) {  
6     if (n == 1L) {  
7         println("1")  
8     } else {  
9         printProduct(n, 2)  
10    }  
11 }
```

ОКОНЧАНИЕ

```
1 fun printProduct(n: Long, from: Long) {  
2     (from .. n).forEach { i ->  
3         when {  
4             n % i == 0L -> {  
5                 println(i)  
6                 printProduct(n / i, i)  
7                 return  
8             }  
9             i * i > n -> {  
10                println(n)  
11                return  
12            }  
13        }  
14    }  
15 }
```