Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Анализ инициализированности в языке Kotlin

Студент гр. 3530901/80201 М.Л. Динмухаметов

Санкт-Петербург
2022

Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Анализ инициализированности в языке Kotlin

по направлению 09.03.01 «Информатика и вычислительная техника»
по образовательной программе
09.03.01_02 «Технологии разработки программного обеспечения»

Выполнил студент гр. 3530901/80201 _____ М.Л. Динмухаметов

Научный руководитель,
ст. преп. _____ М.Х. Ахин

Консультант по нормоконтролю,
к. т. н., доц. _____ А.Г. Новопашенный

Санкт-Петербург
2022

РЕФЕРАТ

На 27 с., 1 рисунков, 3 таблицы

БЕЗОПАСНАЯ ИНИЦИАЛИЗАЦИЯ, KOTLIN, СИСТЕМА ТИПОВ И ЭФФЕКТОВ

Тема выпускной квалификационной работы: «Анализ инициализированности в языке Kotlin».

Данная работа посвящена созданию анализа для нахождения ошибок инициализации в языке Kotlin на основе подхода представленного для Scala 3 [6] . В ходе данной работы решались следующие задачи:

- Анализ существующих решений по поиску ошибок инициализации в языках.
- Разработка анализа для нахождения не безопасной инициализации в языке Kotlin.
- Создание прототипа, реализующего данный подход.
- Оценка производительности и тестирование данного подхода.

Анализ основывается на подходе, который был представлен для языка Scala 3 [6] и модифицирует его для работы в языке Kotlin. Анализ основывается на системе типов и эффектов, и способен находить ряд проблем с инициализацией, которые существующий анализ находить не способен.

Должна ли она тут быть и нужно ли вообще упоминать скалу

THE ABSTRACT

27 pages, 1 pictures, 3 tables

SAFE INITIALIZATION, KOTLIN, TYPE-AND-EFFECTS SYSTEM

TODO Hello

СОДЕРЖАНИЕ

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

JSON	JavaScript Object Notation
NPE	NullPointerException — Исключение возникающее, при использовании переменной хранящей null
XML	eXtensible Markup Language
YAML	Yet Another Markup Language

ВВЕДЕНИЕ

Написание безопасных и надежных программ является сложной задачей для программиста, некоторые ошибки можно легко найти на стадии компиляции, но не все. Так, обращение к еще не инициализированным полям является частой причиной возникновения ошибок в объектно-ориентированных языках программирования. Такие ошибки тяжело находить, а поведение программы может становиться не предсказуемым.

Целью данной работы является разработка анализа для нахождения ошибок инициализации для языка Kotlin. Также в этой работе создается прототип системы безопасной инициализации, и производится оценка его производительности.

Работа состоит из 6 разделов. Первый раздел посвящен проблеме. Включает в себя ряд примеров не безопасной инициализации и ряд обязательных требований без которых анализ будет не состоятельным.

Второй раздел включает в себя обзор существующих решений в индустриальных языках и теоретических подходов к решению проблемы безопасной инициализации. Также в данном разделе выделяются достоинства и недостатки каждого из подходов.

Третий раздел включает в себя постановку задачи.

TODO И что-то еще что должен включать этот раздел

Четвертый раздел посвящен дизайну анализа инициализированности. В данном разделе выбирается подход на основе, которого будет создаваться безопасная инициализация для языка Kotlin. Также рассмотрены проблемы, которые пришлось решать, чтобы выбранный подход мог работать для языка Kotlin.

В пятом разделе рассказывается про детали реализации прототипа и объясняются основные идеи анализа.

Шестой раздел включает в себя тестирование написанного прототипа, а также оценку производительности и качества анализа. Рассказывается на каких проектах был запущен прототип и какие результаты он показал.

1. ПРОБЛЕМА

Проблема не полностью инициализированных объектов известна с появления первых объектно-ориентированных языков программирования, поэтому для данной проблемы существует множество решений, но все эти решения имеют свой набор плюсов и минусов.

Ошибки инициализации имеют достаточно простую природу — это доступ к еще не инициализированным полям, но хоть они и имеют достаточно простую природу, находить ошибки инициализации часто являются головной болью для программистов

Давайте рассмотрим ряд ошибок инициализации для лучшего понимания проблемы. В приведенном примере, если создать экземпляр класса *Hello*, то программа аварийно завершится с *NullPointerException*(NPE), так как во время инициализации свойства *nameLength*, свойство *name* еще не существует.

Листинг 1.1. Пример не безопасной инициализации

```
1 class Hello {
2     fun foo() = name
3
4     val nameLength = foo().length
5     val name = "Alice"
6 }
```

Такую ошибку легко увидеть если код состоит из 6 строчек, но если вы разрабатываете большой и сложный класс, то это становится нетривиальной задачей

Следующий пример показывает, что ошибки инициализации могут появиться и при наследовании. Проблема заключается в том, что вначале идет инициализация класса родителя, в данном случае класс *A*. А во время его инициализации будет вызван переопределенный метод *getName*, который обращается к еще не инициализированному свойству *c*. Оно будет определено только во время инициализации

класса *B*.

Листинг 1.2. Пример не безопасной инициализации

```
1  open class A {
2      val a = "Hello"
3      val b = getName().length
4
5      open fun getName() = a
6  }
7
8  class B : A() {
9      val c = "World"
10
11     override fun getName() = c
12 }
```

Приведенный выше пример специфичен для языка Java, но он вызывает проблемы и в языке Kotlin. Давайте перепишем данный пример для демонстрации, того что в языке Kotlin данная ошибка может быть чуть менее очевидна программисту.

Листинг 1.3. Пример не безопасной инициализации

```
1  open class A1 {
2      open val a = "Hello"
3      val b = a.length
4  }
5
6  class B1 : A1() {
7      override val a = "World"
8  }
```

Кажется, что данный пример не должен приводить к *NPE*, так как *a* и *b* правильно инициализируются, после чего вызывается конструктор *B1*, и завершает создание объекта. Но в языке Kotlin данный пример приведет к *NPE*. Для поля автоматически генерируются компилятором геттеры и сеттеры , и во время инициализации класса *A1* будет вызван уже переопределённый геттер, который в свою очередь возвращает еще неинициализированное поле *a* класса *B1*. На самом деле любое практически любое переопределение членов класса родителя

Не уверен что можно использовать такие термины

ИИИ?

может привести к ошибкам. Эта проблема известна, как *проблема хрупкого базового класса* [4].

Также проблемы инициализации могут нести в себе и вложенные класс. Рассмотрим данный пример.

Листинг 1.4. Пример не безопасной инициализации

```
1  class Outer {
2      val i = Inner()
3      val tag = "Hello"
4
5      inner class Inner {
6          val tagLength = tag.length
7      }
8  }
```

Проблема заключается в том, что вложенный класс может иметь доступ к *this* еще не достроенного внешнего класса. Так класс *Inner* имеет доступ к еще не инициализированному свойству *tag*.

Пример ниже показывает еще один пример не тривиальной ошибки инициализации.

Листинг 1.5. Пример не безопасной инициализации

```
1  class C {
2      val b = O.c
3
4      fun hello(): String = "Hello, World!"
5  }
6
7  object O {
8      val c = C()
9      val hello = c.b.hello()
10 }
```

В данном примере во время инициализации свойства *c*, создается экземпляр класса *C*. Он в свою очередь инициализирует свойство *b*, через свойство *c*, которое еще находится в стадии инициализации. В данной работе не рассматривается проблема инициализации статических объектов, но они также могут быть источниками ошибок иници-

ализации, и их поддержка, возможно, будет добавлена в следующих версиях анализа

Проблема ошибок инициализации создает трудности не только для программистов, но и для разработчиков компиляторов и дизайнеров языков программирования, так если в языке нет безопасной инициализации, то нельзя гарантировать неизменяемость или ненулевость [5]

нормальн
слово

Не уверен, что хорошо ссылаться на личный блог, но там и правда неплохо все объяснено ссылка из статьи

Kotlin — это null-безопасный язык, и система типов языка Kotlin гарантирует, что *non-nullable* тип не может быть null.

Листинг 1.6. Пример не безопасной инициализации

```
1 class Message {
2     fun foo(): String = name
3
4     val hello: String = "Hello, " + foo()
5     val name: String = "Alice"
6 }
```

Если попробовать напечатать значение хранящееся в *hello*, то выведется *"Hello, null"*, хотя *name* не может быть null, потому что это гарантирует система типов языка Kotlin. Это делает ошибки инициализации в языке Kotlin менее очевидными и ожидаемые

1.1. Резюме

В данном разделе была рассмотрена причина возникновения ошибок инициализации. Они возникают при обращении к еще не инициализированным полям. А источником ошибок инициализации могут быть:

- методы
- внутренние классы

- наследование
- статические объекты
- итд

В языке Kotlin такие ошибки могут быть менее ожидаемыми, так как это null-безопасный язык. Также, отмечается, что ошибки инициализации это проблема не только для программистов, но и для дизайнеров языка.

2. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Может показаться, что достаточно запретить использование не до конца сконструированных значений, иначе говоря запретить использование *this* в конструкторе, но это не так.

Листинг 2.1. Пример не безопасной инициализации

```
1 class Parent {
2     val tag = "Hello"
3 }
4
5 class Child(parent: Parent) {
6     val tag = parent.tag
7 }
```

2.1. Безопасная инициализация в языке Kotlin

TODO Примеры как легко сломать и что по сути безопасной инициала

2.2. Инициализации в языке Swift

TODO Рассказ про инициализацию в Swift

2.3. Инициализации в Dart

Если хватит времени рассмотреть инициализацию в Dart

2.4. Маскирующие типы

TODO Маскирующие типы и какие проблемы в себе несут

2.5. The freedom model

Нормальный перевод "Модель свободы"?

TODO Рассказ про freedom model и плавный переход к решению из скалы

2.6. Безопасная инициализация для языка Scala 3

TODO Рассказ про безопасную инициализацию в скале примеры что может анализ и плавный переход к тому почему используем

2.7. Резюме

TODO Резюме про то что существует два классических подхода что безопасная инициализация может обеспечиваться самим языком или быть чем-то внешним никак не ограничивая синтаксис

3. ПОСТАНОВКА ЗАДАЧИ

TODO Не очень понимаю что здесь нужно написать. Вроде про то что проблема есть в котлине и какими способами ее можно решить мы уже обсудили в предыдущем разделе

4. ДИЗАЙН

Придумать нормальное название разделу

4.1. Выбор подхода для создания безопасной инициализации в языке Kotlin

TODO Говорим какой подход выбрали почему не выбрали другие итд

4.2. Проблемы подхода для языка Kotlin

TODO Рассказ про проблемы которые существуют для применения решения из скалы и их решение

4.3. Резюме

TODO Резюме про выбранный подход и набор проблем

5. СОЗДАНИЕ ПРОТОТИПА

Раздел может быть различного размера для начала опишем минимум

5.1. Система типов и эффектов

TODO Рассказываем про итоговую систему типов и эффектов, что она сильно не изменилась и взята из скалы

5.2. Интеграция в компилятор

TODO Рассказываем про то как это все взаимодействует с компилятором

5.3. Резюме

TODO Кратко как это работает в компиляторе

6. ТЕСТИРОВАНИЕ И ОЦЕНКА РЕЗУЛЬТАТОВ

6.1. Тестирование

TODO Тестирование на чем запускались итд

6.2. Оценка результатов

TODO Производительность и количество ворнингов

6.3. Резюме

TODO Что получилось

7. TMP

TODO Проблема с примерами

7.1. TMP-1

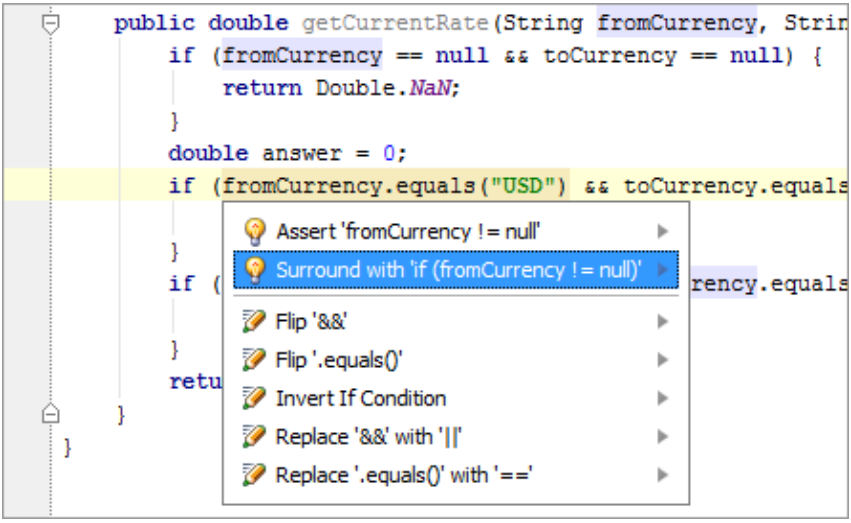

Рис.7.1. Рекомендации по проведению исследований в рамках диссертации

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin.

Таблица 7.1

Example

	LLVM IR		PS		AI	
	SAT	UNSAT	SAT	UNSAT	SAT	UNSAT
beanstalkd	356	252	360	161	360	247
clib	599	258	960	234	960	449

Таблица 7.3

Решетка замечательности аббревиатур

XML < JSON < YAML

Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

helloтабл. 7.1 It is of great importance that you use correct references in your dissertation2.2 esent studies show that it can increase the chances of successful defense by as much as 3,17 percent [1–3].

Таблица 7.2

Название таблицы

top left	top right
bot left	bot right

You can use all kinds of abbreviations that don't mean anything, but add a false sense of importance and significance to your work. Some of these abbreviations are:

- eXtensible Markup Language (XML)
- JavaScript Object Notation (JSON)
- Yet Another Markup Language (YAML)

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales

commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam

lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

ЗАКЛЮЧЕНИЕ

TODO Ну заключение

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Василий Пупкин, Василиса Пупкина, Пупкинский В.П. Русская статья // Научно-технические ведомости НИИ ЧАВО. — 2000. — № 16. — С. 32–64.
2. Нимейер П., Леук Д. Программирование на Java. — Эксмо, 2014. — ISBN: 9785457661004. — URL: https://books.google.ru/books?id=zVpUBQAAQBAJ.
3. ANTLR [Электронный ресурс], Веб-сайт проекта ANTLR. — URL: http://www.antlr.org/ (дата обращения: 18.06.2015).
4. Mikhajlov Leonid, Sekerinski Emil. A study of the fragile base class problem // ECOOP'98 — Object-Oriented Programming / Ed. by Eric Jul. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1998. — P. 355–382.
5. On partially-constructed objects [Электронный ресурс], Личный блог Джо Даффи. — URL: http://joeduffyblog.com/2010/06/27/on-partiallyconstructed-objects/ (дата обращения: 18.05.2022).
6. A Type-and-Effect System for Object Initialization / Fengyun Liu, Ondřej Lhoták, Aggelos Biboudis et al. // Proc. ACM Program. Lang. — 2020. — nov. — Vol. 4, no. OOPSLA. — 28 p. — URL: https://doi.org/10.1145/3428243.

Приложение

ЛИСТИНГИ

Листинг 1. Код на Java

```
class Hello {
    fun foo() = name

    val nameLength = foo().length
    val name = "Alice"
}
```