Rust on Android?

How to write bugs once and ship them many times!

About me

You can find me:

- on Twitter: @BitPogo
- on Github: bitPogo
- on Kotlin Slack, Rust Slack or ADG Berlin Slack

And I am currently looking for a new Job*...

About today

What I am going to do:

- a (very very) short Intro into Rust
- a (very very) brief how-to bind Rust to Kotlin
- show you some Code (later)
- on board you on the idea
  (so I can make more Rust talks)

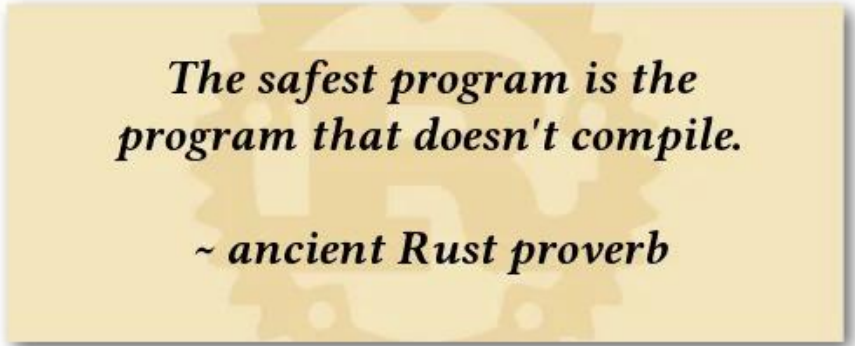
Why Rust? (Is Kotlin not enough?)

- You need extended Crossplatform capabilities
- You have to minimize overhead
- You need native Libraries (like ML)
- You want to write yet another Gradle Plugin
- …
- or whenever you want to use the NDK

But most importantly - because we can!!!!

What addresses Rust?

- introduced by Mozilla (first stable 2015), since 2021 owned by the Rust Foundation (because of COVID-19)
- designed to replace C/C++, while incremental expanding Rust in Firefox
- zero abstraction overhead, while being memorysafe
- modern, small flexible and free
- immutable types by default (in opposite to C/C++)
- UTF-8 support
- multithreading while guaranteeing safe/fearless concurrency
- type safety
- ....



Keyconcepts

1. Borrowing:
    - the compiler keeps track of the ownership of **mutable** references
    - scope dependent
    - the compiler does the (de-)allocation for you
    - you can use fearless threads (it even prevents race conditions)
    - But:
        - the Borrow checker is dumb
        - understanding how it works in real is a road of pain (use your Juniors for that!)

Keyconcepts

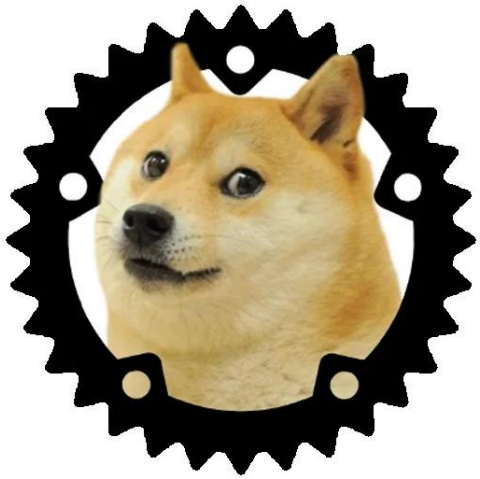
2. Lifetimes:
    - every variable has a lifetime (explicit or implicit)
    - binds a variable/parameter/field to its scope

3. Ownership:
    - there can be only one (Highlander rule)
    - move/borrow/copy semantic

…and

it feels familiar:

- struct (like in Swift)
- they is simple visibility (but only private and public)
- there is no inheritance (no interfaces*), but traits
- there is no polymorphism*, but traits and Generics (pragmatic/ad-hoc polymorphism)
- consistent control flow (via Monads and panic)
- structured coconcurrency (aka Coroutines)
- ...



Rust with Kotlin - what does it takes?

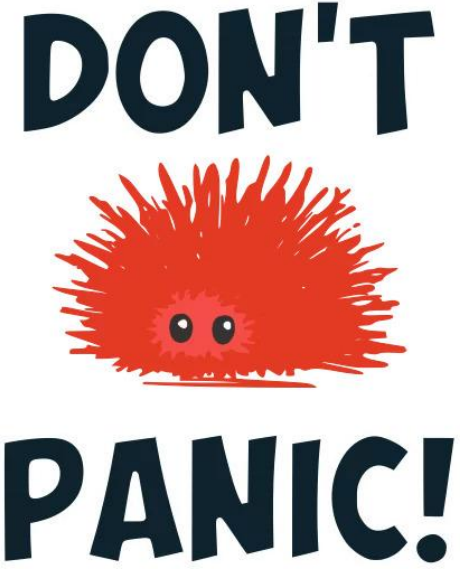
- Flapigen
- JNI
- Mozilla's Gradle Plugin
- Stardust's Gradle Plugin
- NDK Crate
- Rustup
- Memes
- RustOnRails
- ProtoBuf
- ...

...and of you!!!

Pitfalls

- Interactions between Kotlin and Rust
  - it slows down (due to Memory Management)
  - hard to debug
  - Kotlin Concurrency <-> Rust Concurrency
- Error Handling
- Complex Data Types
- Async Rust (see Tokio)
- ...

=> Keep each part as much as possible self contained
=> Keep the interaction model simple
=> use Atomics and Strings (with ProtoBuf) for transfering Data



Demo time....

Questions?