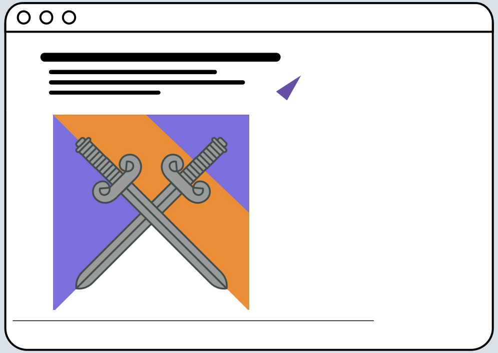
KOTLINX SERIALIZATION

How to make your own custom serialization library



ABOUT ME

Andrey Kuleshov

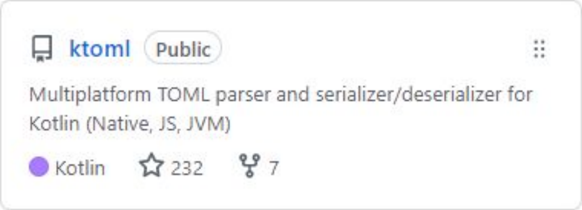
https://github.com/**akuleshov7**

awesome kotlin

intel Deutsche Bank HUAWEI

[KT]oml

powered by kotlinx.serialization



ktoml Public

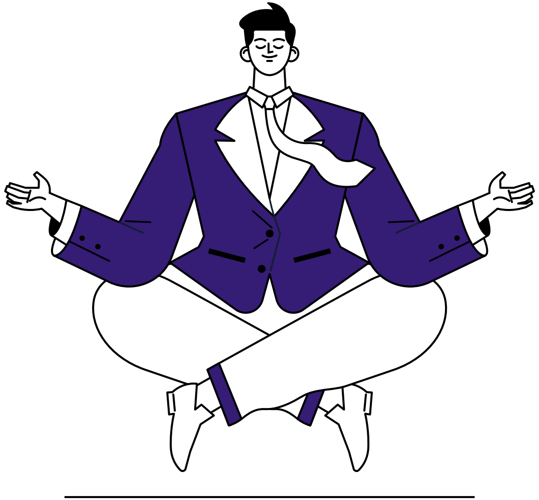
Multiplatform TOML parser and serializer/deserializer for Kotlin (Native, JS, JVM)

● Kotlin ☆ 232 7

LET'S VOTE FIRST!

What is the most friendly format for you to configure your app?

1) YAML
2) JSON
3) TOML
4) Properties
5) XML
6) Other



MOTIVATION OF THIS TALK

- Tons of question in Slack and Telegram chats

MOTIVATION OF THIS TALK

- Tons of question in Slack and Telegram chats
- Love the design and want to share

MOTIVATION OF THIS TALK

- Tons of question in Slack and Telegram chats
- Love the design and want to share
- Elegant and so aesthetic!

MOTIVATION OF THIS TALK

- Tons of question in Slack and Telegram chats
- Love the design and want to share
- Elegant and so aesthetic!
- Useful for users and authors of libraries

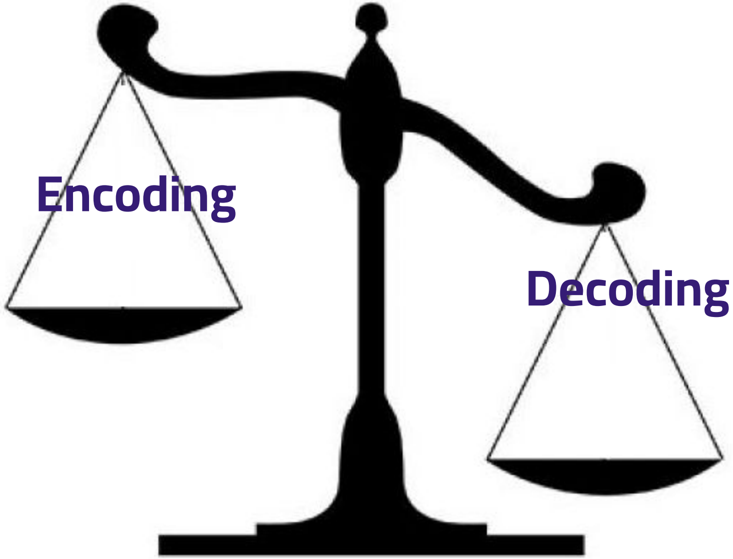
MOTIVATION OF THIS TALK

- Tons of question in Slack and Telegram chats
- Love the design and want to share
- Elegant and so aesthetic!
- Useful for users and authors of libraries
- 20 formats - you can participate in dev

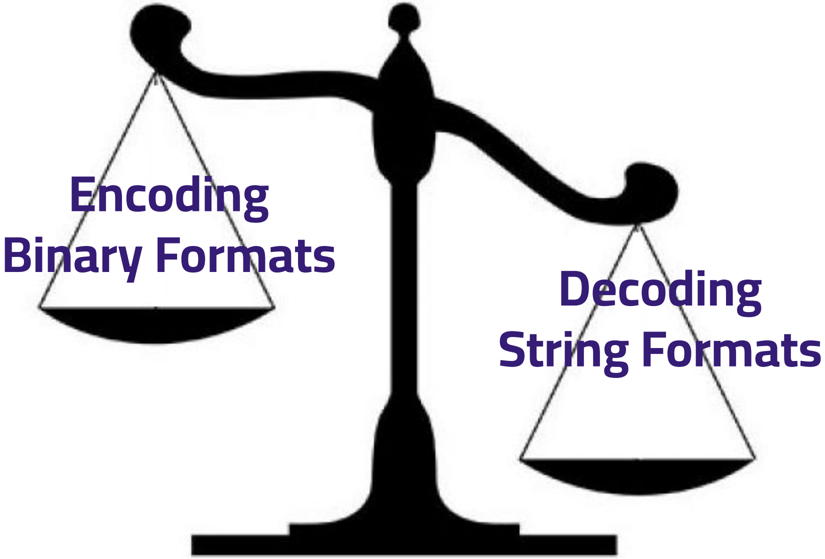
ELEGANCY ❤️

```kotlin
import kotlinx.serialization.Serializable
import kotlinx.serialization.encodeToString

@Serializable
data class Credentials(
    val login: String,
    val password: String
)

val data = Credentials("akuleshov7", "qwerty")

Json.encodeToString(data)
Toml.encodeToString(data)
Yaml.encodeToString(data)
```

SCOPE



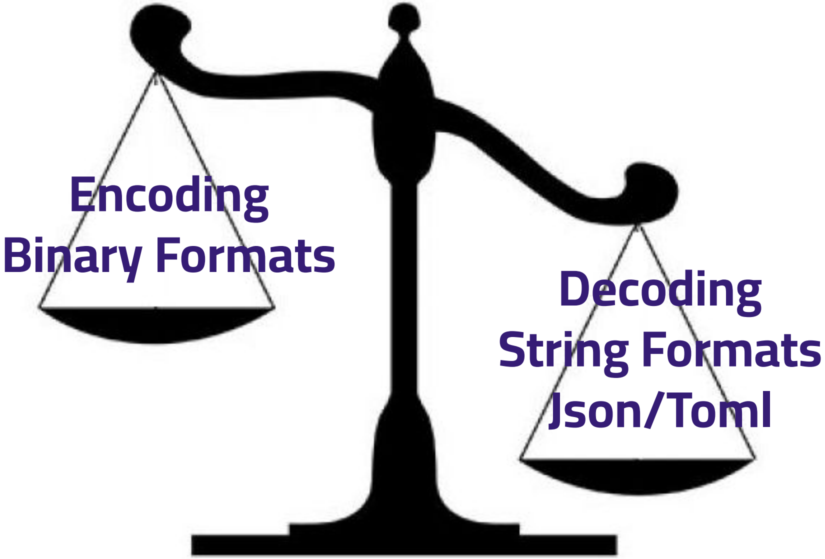
SCOPE



Encoding
Binary Formats

Decoding
String Formats

SCOPE



Encoding
Binary Formats

Decoding
String Formats
Json/Toml

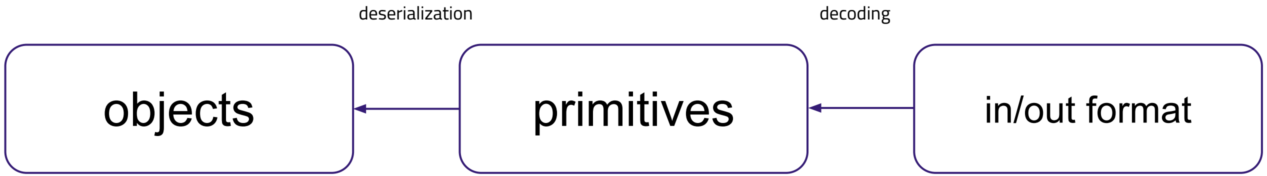
LET'S DEFINE TERMS

objects

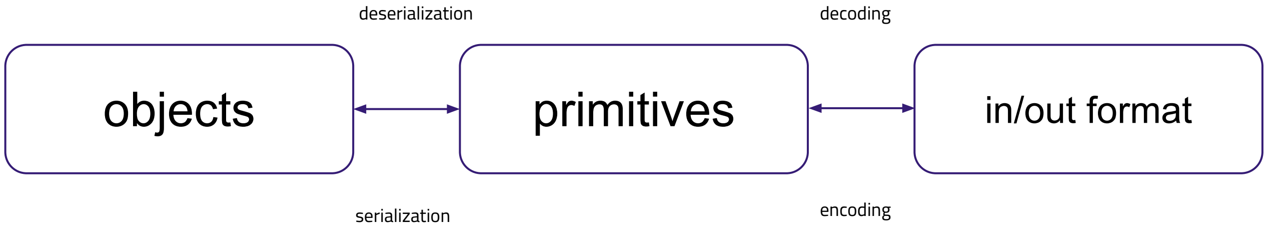
primitives

in/out format

LET'S DEFINE TERMS



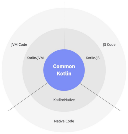
LET'S DEFINE TERMS



Serialization is **decoupled** from the encoding process to make it **format-agnostic**

BRIEFLY

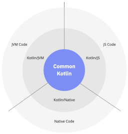
KOTLINX.SERIALIZATION



Is a **Multiplatform** library

BRIEFLY

KOTLINX.SERIALIZATION

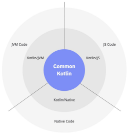


Is a **Multiplatform** library

Is a Compiler plugin (> K1.4)

BRIEFLY

KOTLINX.SERIALIZATION



Is a **Multiplatform** library

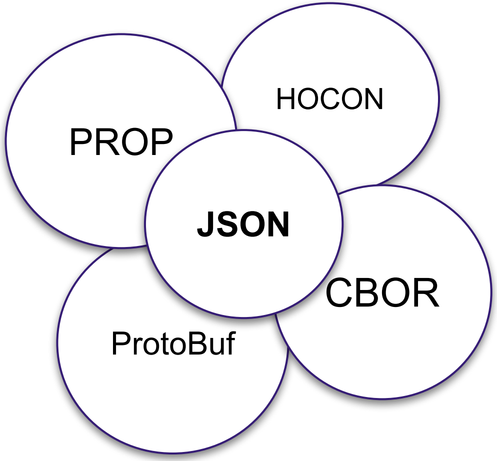
Is a Compiler plugin (> K1.4)

Has it's own Gradle/Maven plugins

BRIEFLY

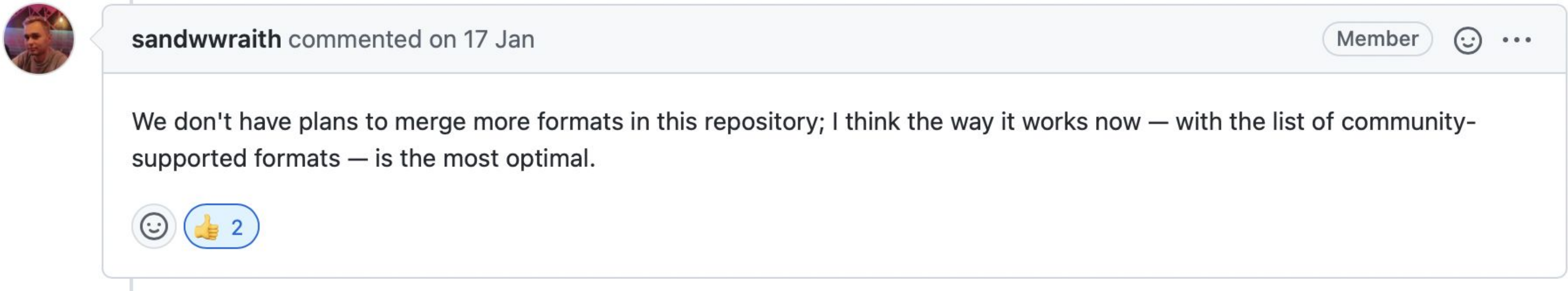
KOTLINX.SERIALIZATION

CORE



BRIEFLY

KOTLINX.SERIALIZATION

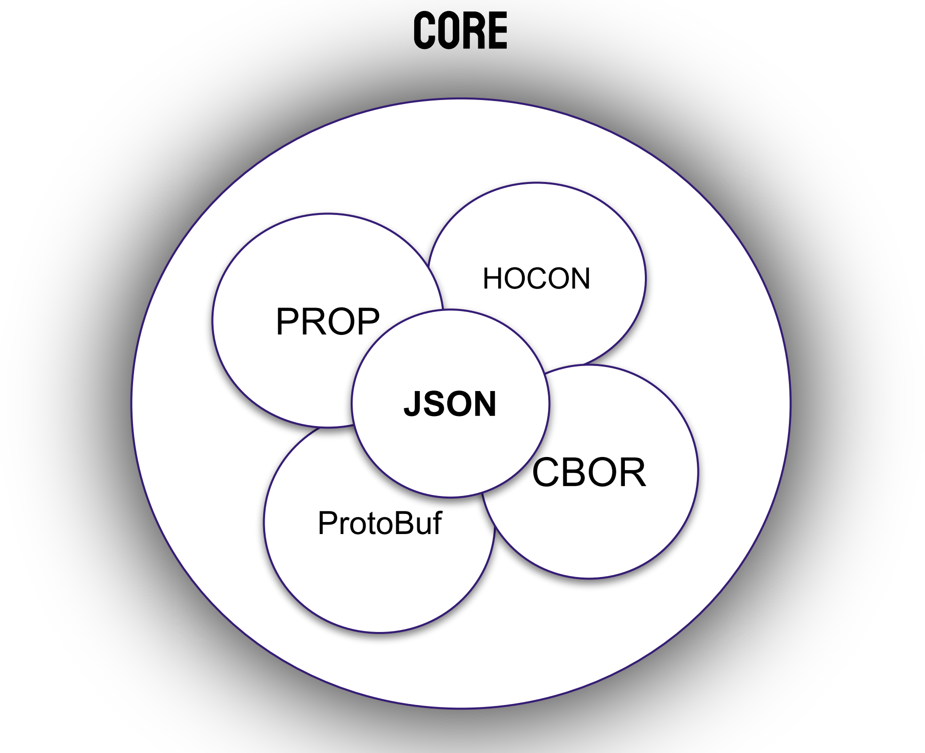


sandwwraith commented on 17 Jan

We don't have plans to merge more formats in this repository; I think the way it works now — with the list of community-supported formats — is the most optimal.

BRIEFLY

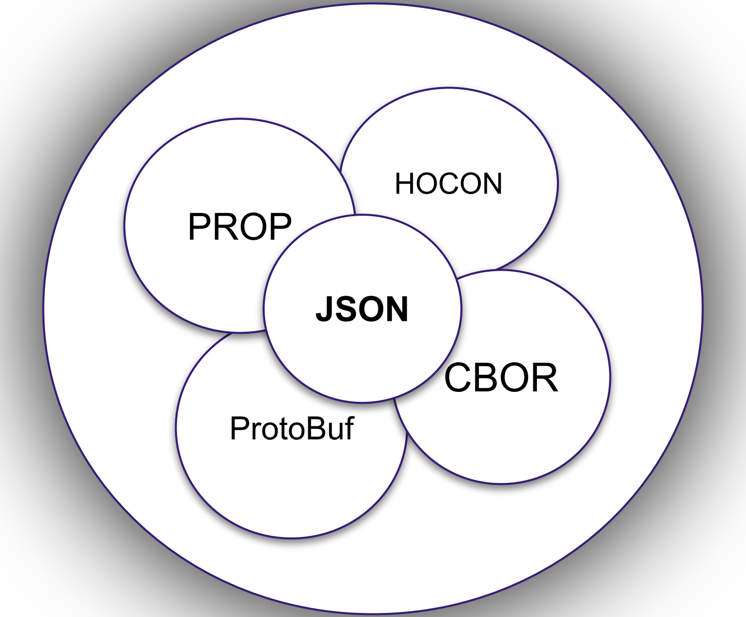
KOTLINX.SERIALIZATION



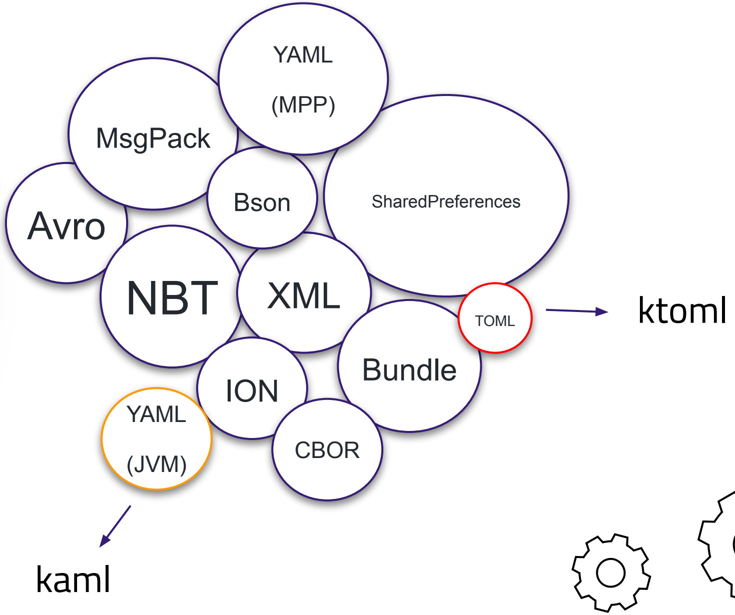
BRIEFLY

KOTLINX.SERIALIZATION

CORE



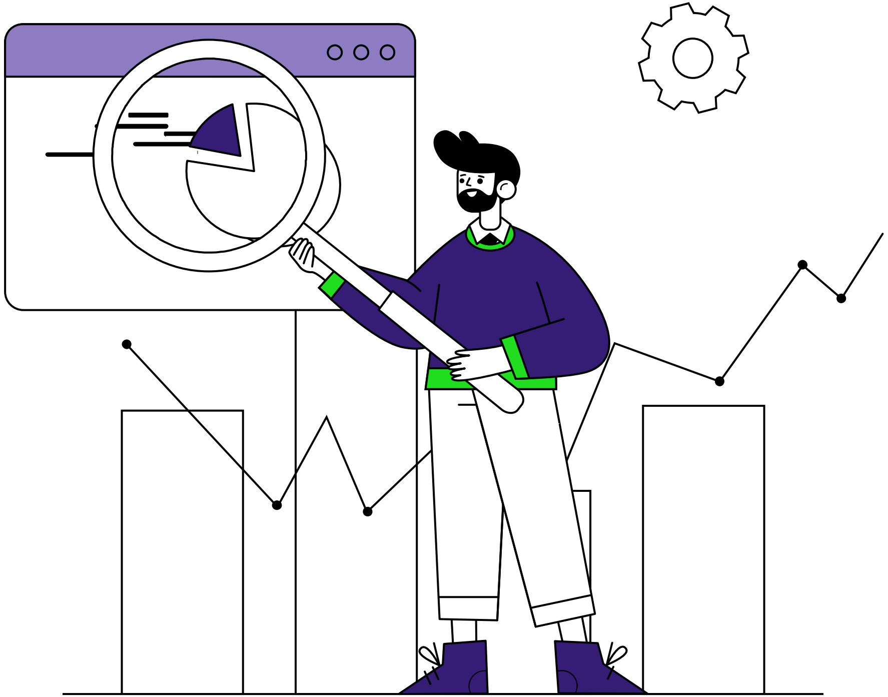
COMMUNITY



01

USER SCENARIOS

Several notes for users of serialization libraries

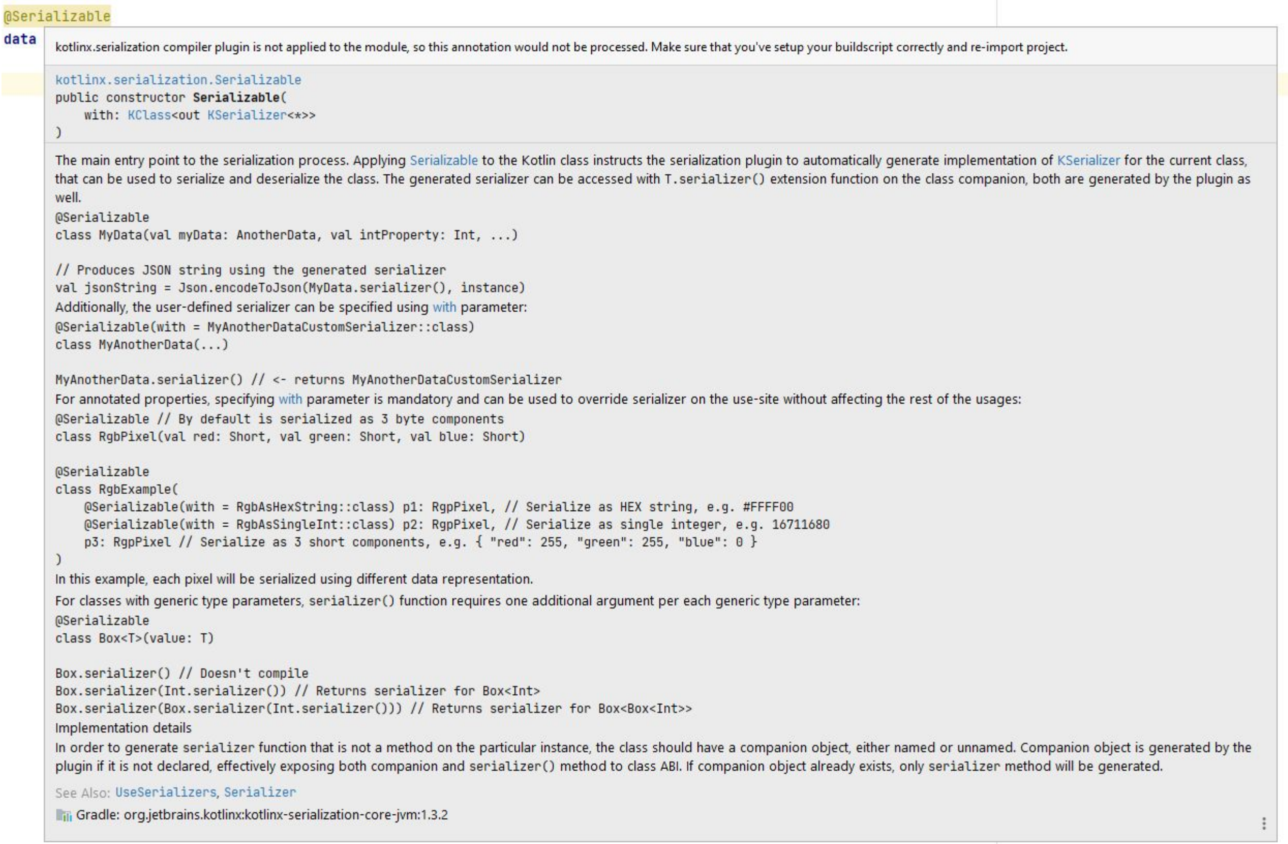


```
plugins {
    kotlin("plugin.serialization") version "1.6.20"
}
```

jetbrains/kotlin

kotlin-serialization/**kotlin-serialization-compiler**

```
plugins {
    kotlin("plugin.serialization") version "1.6.20"
}
```



```
@Serializable
data
```

kotlinx.serialization compiler plugin is not applied to the module, so this annotation would not be processed. Make sure that you've setup your buildscript correctly and re-import project.

```
kotlinx.serialization.Serializable
public constructor Serializable(
    with: KClass<out KSerializer<*>>
)
```

The main entry point to the serialization process. Applying Serializable to the Kotlin class instructs the serialization plugin to automatically generate implementation of KSerializer for the current class, that can be used to serialize and deserialize the class. The generated serializer can be accessed with `T.serializer()` extension function on the class companion, both are generated by the plugin as well.

```
@Serializable
class MyData(val myData: AnotherData, val intProperty: Int, ...)

// Produces JSON string using the generated serializer
val jsonString = Json.encodeToJson(MyData.serializer(), instance)
```

Additionally, the user-defined serializer can be specified using with parameter:

```
@Serializable(with = MyAnotherDataCustomSerializer::class)
class MyAnotherData(...)

MyAnotherData.serializer() // <- returns MyAnotherDataCustomSerializer
```

For annotated properties, specifying with parameter is mandatory and can be used to override serializer on the use-site without affecting the rest of the usages:

```
@Serializable // By default is serialized as 3 byte components
class RgbPixel(val red: Short, val green: Short, val blue: Short)

@Serializable
class RgbExample(
    @Serializable(with = RgbAsHexString::class) p1: RgpPixel, // Serialize as HEX string, e.g. #FFFF00
    @Serializable(with = RgbAsSingleInt::class) p2: RgpPixel, // Serialize as single integer, e.g. 16711680
    p3: RgpPixel // Serialize as 3 short components, e.g. { "red": 255, "green": 255, "blue": 0 }
)
```

In this example, each pixel will be serialized using different data representation.

For classes with generic type parameters, `serializer()` function requires one additional argument per each generic type parameter:

```
@Serializable
class Box<T>(value: T)

Box.serializer() // Doesn't compile
Box.serializer(Int.serializer()) // Returns serializer for Box<Int>
Box.serializer(Box.serializer(Int.serializer())) // Returns serializer for Box<Box<Int>>
```

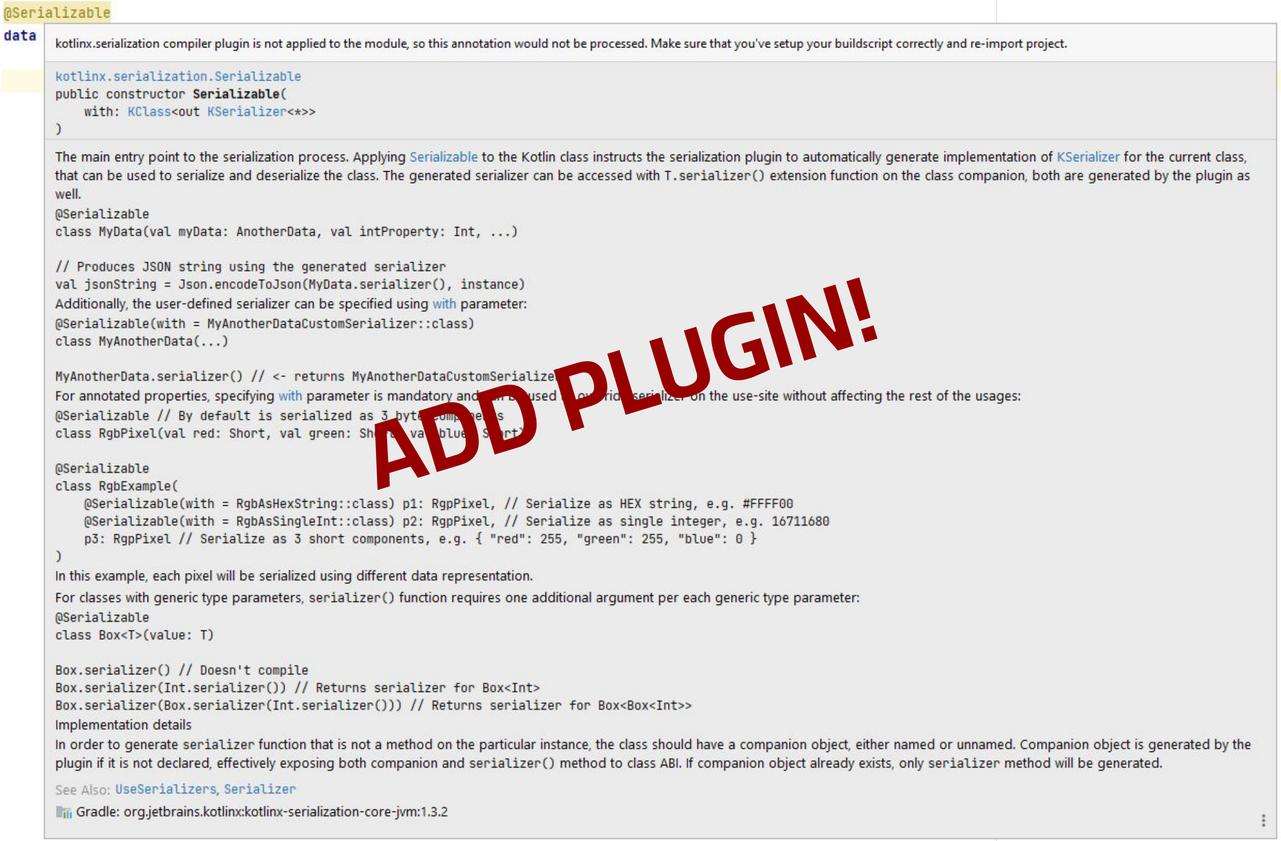
Implementation details

In order to generate `serializer` function that is not a method on the particular instance, the class should have a companion object, either named or unnamed. Companion object is generated by the plugin if it is not declared, effectively exposing both companion and `serializer()` method to class ABI. If companion object already exists, only `serializer` method will be generated.

See Also: UseSerializers, Serializer

Gradle: org.jetbrains.kotlinx:kotlinx-serialization-core-jvm:1.3.2

```kotlin
plugins {
    kotlin("plugin.serialization") version "1.6.20"
}
```



```kotlin
@Serializable
data
```

kotlinx.serialization compiler plugin is not applied to the module, so this annotation would not be processed. Make sure that you've setup your buildscript correctly and re-import project.

```
kotlinx.serialization.Serializable
public constructor Serializable(
    with: KClass<out KSerializer<*>>
)
```

The main entry point to the serialization process. Applying Serializable to the Kotlin class instructs the serialization plugin to automatically generate implementation of KSerializer for the current class, that can be used to serialize and deserialize the class. The generated serializer can be accessed with `T.serializer()` extension function on the class companion, both are generated by the plugin as well.

```
@Serializable
class MyData(val myData: AnotherData, val intProperty: Int, ...)

// Produces JSON string using the generated serializer
val jsonString = Json.encodeToJson(MyData.serializer(), instance)
```

Additionally, the user-defined serializer can be specified using with parameter:

```
@Serializable(with = MyAnotherDataCustomSerializer::class)
class MyAnotherData(...)

MyAnotherData.serializer() // <- returns MyAnotherDataCustomSerializer
```

For annotated properties, specifying with parameter is mandatory and can be used to override serializer on the use-site without affecting the rest of the usages:

```
@Serializable // By default is serialized as 3 byte components
class RgbPixel(val red: Short, val green: Short, val blue: Short)

@Serializable
class RgbExample(
    @Serializable(with = RgbAsHexString::class) p1: RgpPixel, // Serialize as HEX string, e.g. #FFFF00
    @Serializable(with = RgbAsSingleInt::class) p2: RgpPixel, // Serialize as single integer, e.g. 16711680
    p3: RgpPixel // Serialize as 3 short components, e.g. { "red": 255, "green": 255, "blue": 0 }
)
```

In this example, each pixel will be serialized using different data representation.

For classes with generic type parameters, `serializer()` function requires one additional argument per each generic type parameter:

```
@Serializable
class Box<T>(value: T)

Box.serializer() // Doesn't compile
Box.serializer(Int.serializer()) // Returns serializer for Box<Int>
Box.serializer(Box.serializer(Int.serializer())) // Returns serializer for Box<Box<Int>>
```

Implementation details

In order to generate `serializer` function that is not a method on the particular instance, the class should have a companion object, either named or unnamed. Companion object is generated by the plugin if it is not declared, effectively exposing both companion and `serializer()` method to class ABI. If companion object already exists, only `serializer` method will be generated.

See Also: UseSerializers, Serializer

Gradle: org.jetbrains.kotlinx:kotlinx-serialization-core-jvm:1.3.2

ADD PLUGIN!

SIMPLE EXAMPLE OF USAGE

```
import kotlinx.serialization.Serializable
import kotlinx.serialization.decodeFromString
import kotlinx.serialization.json.Json

@Serializable
data class Credentials(
    val login: String,
    val password: String
)

fun main() {
   val str =
       """
           { "login": "akuleshov7", "password": "qwerty" }
       """

   Json.decodeFromString<Credentials>(str)
}
```

SIMPLE EXAMPLE OF USAGE

```
import kotlinx.serialization.Serializable
import kotlinx.serialization.decodeFromString
import kotlinx.serialization.json.Json

@Serializable
data class Credentials(
     val login: String,
     val password: String
)

fun main() {
    val str =
        """
            { "login": "akuleshov7", "password": "qwerty" }
        """

    Json.decodeFromString<Credentials>(str)
}
```

SERIALIZATION FEATURES

OPTIONAL DATA

```kotlin
@Serializable
data class Credentials(
    val login: String = "asm0di0",
    val password: String
)
```

```json
{ "password": "qwerty" }
```

SERIALIZATION FEATURES

OPTIONAL DATA

```
@Serializable
data class Credentials(
    val login: String = "asm0di0",
    val password: String
)
```

(!) Including calculated values

```
{ "password": "qwerty" }
```

SERIALIZATION FEATURES

NULLABILITY

```
{
     "login": null,
     "password": "qwerty"
}
```

```
@Serializable
data class Credentials(
    val login: String?,
    val password: String
)
```

(!) Type safe

SERIALIZATION FEATURES

REQUIRED FIELDS

```
@Serializable
data class Credentials(
    val login: String,

    val password: String = "optional"
)
```

SERIALIZATION FEATURES

REQUIRED FIELDS

```
@Serializable
data class Credentials(
    val login: String,
    @Required
    val password: String = "optional"
)
```

SERIALIZATION FEATURES

REQUIRED FIELDS

```
@Serializable
data class Credentials(
    val login: String,
    @Required
    val password: String = "optional"
)
```

TRANSIENT FIELDS

```
@Serializable
data class Credentials(
    @Transient
    val login: String
)
```

SERIALIZATION FEATURES

REQUIRED FIELDS

```
@Serializable
data class Credentials(
    val login: String,
    @Required
    val password: String = "optional"
)
```

TRANSIENT FIELDS

```
@Serializable
data class Credentials(
    @Transient
    val login: String
)
```

(!) In most deserializers unknown field will cause an error. For example:

Kaml: strictMode
Ktoml: ignoreUnknownNames

MINOR NOTES

- Default values are **not** encoded (by the default in most serializers) `@EncodeDefault` - to avoid it

MINOR NOTES

- Default values are **not** encoded (by the default in most serializers) `@EncodeDefault` - to avoid it
- All parameters of the class primary constructors should be properties (val/var)

MINOR NOTES

- Default values are **not** encoded (by the default in most serializers) `@EncodeDefault` - to avoid it
- All parameters of the class primary constructors should be properties (val/var)
- Generics are supported out of the box

MINOR NOTES

- Default values are **not** encoded (by the default in most serializers) @EncodeDefault - to avoid it
- All parameters of the class primary constructors should be properties (val/var)
- Generics are supported out of the box
- SerialName annotation (even for enum entries):

```
@SerialName("my-serial-name")
val name: Long
```

BENEFITS

MULTIPLATFORM

Supported in KotlinJS, Native, JVM, etc.

But you can also serialize 3rd party

BENEFITS

MULTIPLATFORM

Supported in KotlinJS, Native, JVM, etc.

But you can also serialize 3rd party

DATA VALIDATION

```
@Serializable
data class Credentials(
    val password: String
) {
   init {
       require(password ≠ "qwerty")
   }
}
```

BENEFITS

MULTIPLATFORM

Supported in KotlinJS, Native, JVM, etc.

But you can also serialize 3rd party

DATA VALIDATION

```
@Serializable
data class Credentials(
    val password: String
) {
   init {
       require(password ≠ "qwerty")
   }
}
```

COMPILER PLUGIN

Useful codegen during the compilation.

NO REFLECTION

02

INSPECTING THE CORE

GETTING INTO THE INSIDES

NYPD
Encoder
Decoder

BASICS: KSERIALIZER

- Compiler plugin generates an instance of the `KSerializer` interface for every **`@Serializable`** class

BASICS: KSERIALIZER

- Compiler plugin generates an instance of the `KSerializer` interface for every **`@Serializable`** class
- This class has **`.serializer()`** method to return this `KSerializer`

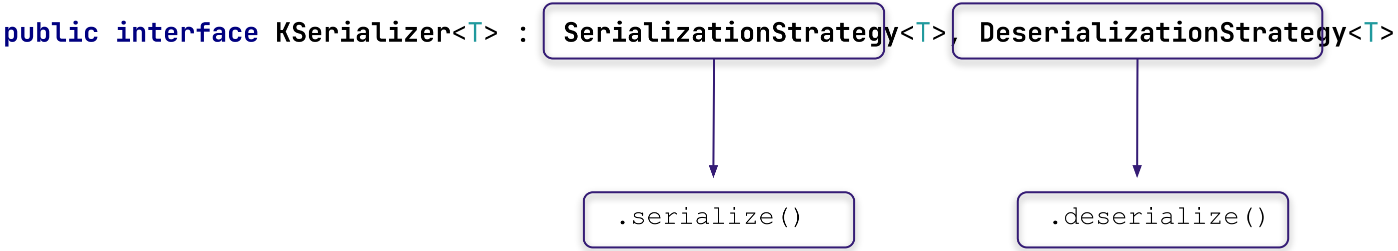
BASICS: KSERIALIZER

- Compiler plugin generates an instance of the `KSerializer` interface for every **`@Serializable`** class
- This class has **`.serializer()`** method to return this `KSerializer`

```
public interface KSerializer<T> :  SerializationStrategy<T>, DeserializationStrategy<T>
```

BASICS: KSERIALIZER

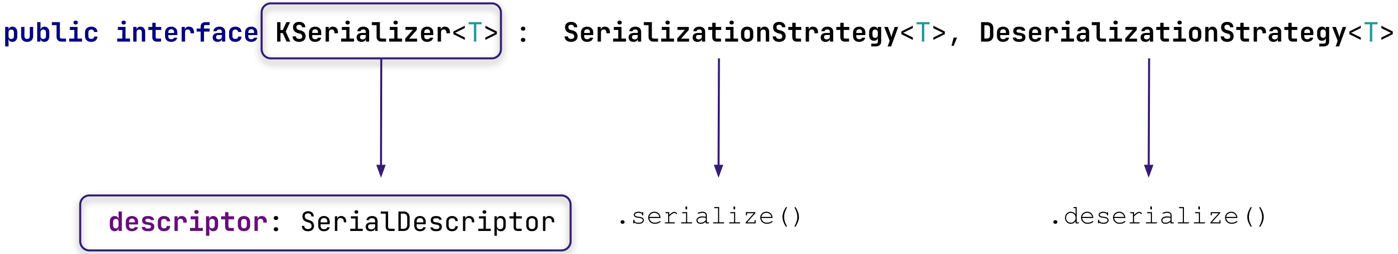
- Compiler plugin generates an instance of the `KSerializer` interface for every **`@Serializable`** class
- This class has **`.serializer()`** method to return this `KSerializer`



```
public interface KSerializer<T> : SerializationStrategy<T>, DeserializationStrategy<T>
```

BASICS: KSERIALIZER

- Compiler plugin generates an instance of the `KSerializer` interface for every `@Serializable` class
- This class has `.serializer()` method to return this `KSerializer`



```
public interface KSerializer<T> :  SerializationStrategy<T>, DeserializationStrategy<T>
```

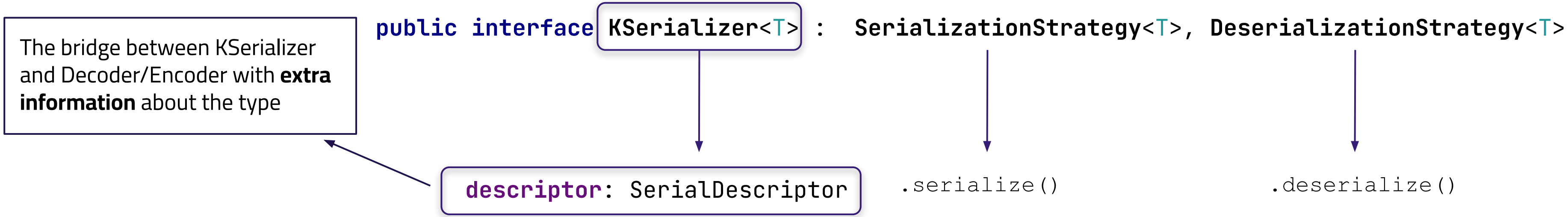
```
descriptor: SerialDescriptor
```

```
.serialize()
```

```
.deserialize()
```

BASICS: KSERIALIZER

- Compiler plugin generates an instance of the `KSerializer` interface for every **`@Serializable`** class
- This class has **`.serializer()`** method to return this `KSerializer`



```
public interface KSerializer<T> : SerializationStrategy<T>, DeserializationStrategy<T>
```

```
descriptor: SerialDescriptor
```

```
.serialize()
```

```
.deserialize()
```

The bridge between KSerializer and Decoder/Encoder with **extra information** about the type

WHAT KIND OF INFORMATION?

```kotlin
data class Credentials(
    val login: String,
    val password: String
)
```

NAME	INDEX
login	0
password	1

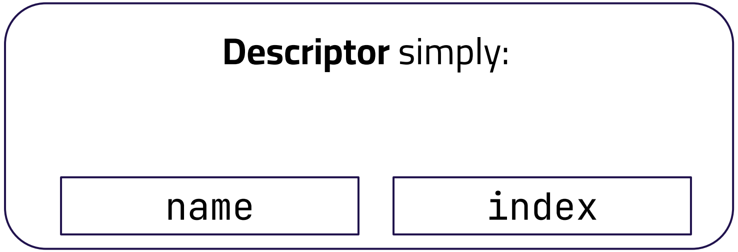
WHAT KIND OF INFORMATION?

```
data class Credentials(
    val login: String,
    val password: String
)
```

NAME	INDEX
login	0
password	1

+ metadata

WHAT KIND OF INFORMATION?



Descriptor simply:

name

index

```
fun getElementName(index: Int)
```

```
fun getElementIndex(name: String)
```

```
fun getElementDescriptor(index: Int)
```

LOVELY EXAMPLE AGAIN

```kotlin
import kotlinx.serialization.Serializable
import kotlinx.serialization.decodeFromString
import kotlinx.serialization.json.Json

@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```kotlin
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
    val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
    var login: String? = null
    var password: String? = null
    val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
        while (true) {
            when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
                -1 -> break
                0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
                1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
            }
        }
    return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
   val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
   var login: String? = null
   var password: String? = null
   val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
       while (true) {
           when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
               -1 -> break
               0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
               1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
           }
       }
   return Credentials(login!!, password!!)
}
```

PSEUDOCODE FOR DESERIALIZER

```
fun deserialize(decoder: Decoder): Credentials {
    val descriptorOfCredentialsClass: SerialDescriptor = this.getDescriptor()
    var login: String? = null
    var password: String? = null
    val compositeDecoder = decoder.beginStructure(descriptorOfCredentialsClass)
        while (true) {
            when (compositeDecoder.decodeElementIndex(descriptorOfCredentialsClass)) {
                -1 -> break
                0 -> login = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 0)
                1 -> password = compositeDecoder.decodeStringElement(descriptorOfCredentialsClass, 1)
            }
        }
    return Credentials(login!!, password!!)
}
```

LOVELY EXAMPLE AGAIN

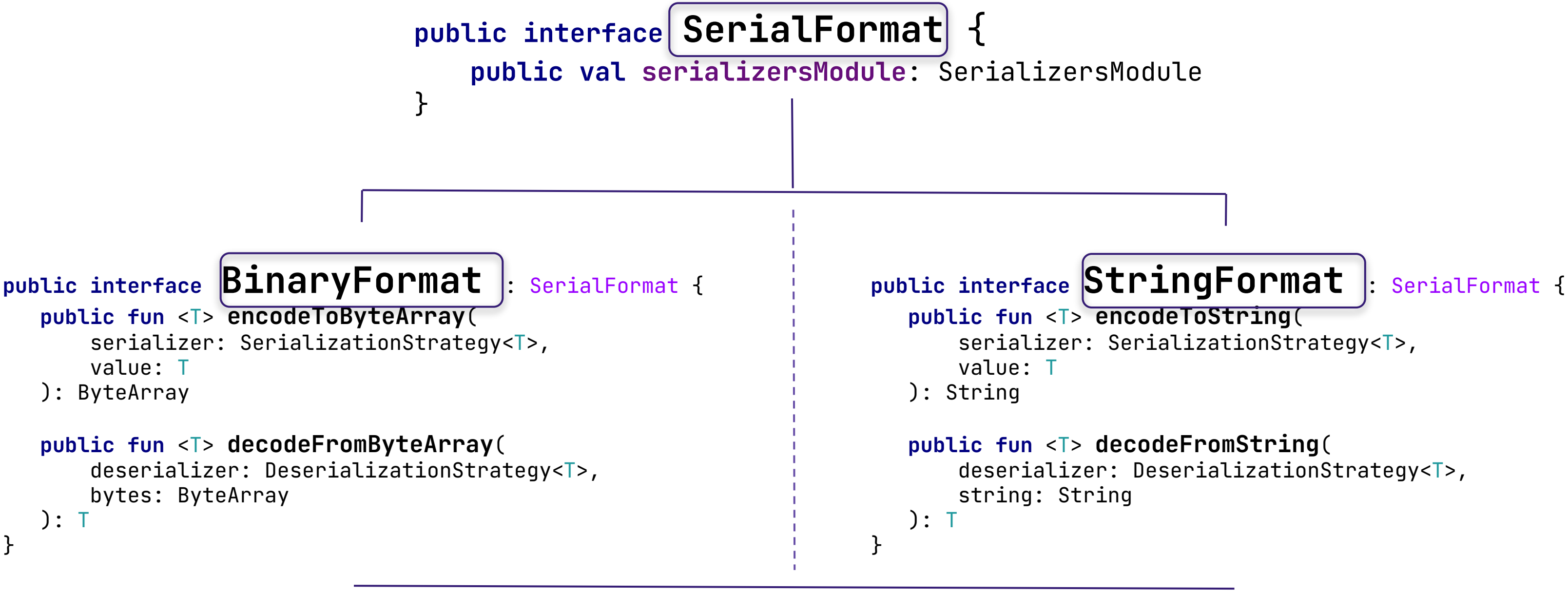
```kotlin
import kotlinx.serialization.Serializable
import kotlinx.serialization.decodeFromString
import kotlinx.serialization.json.Json

@Serializable
data class Credentials(
    val login: String,
    val password: String
)

fun main() {
   Json.decodeFromString<Credentials>("")
}
```

SERIAL FORMAT

```
public interface SerialFormat {
    public val serializersModule: SerializersModule
}
```

```
public inline fun <reified T> StringFormat.decodeFromString(string: String): T =
    decodeFromString(serializersModule.serializer<T>(), string)
```

SERIAL FORMAT



```
public interface SerialFormat {
    public val serializersModule: SerializersModule
}
```

```
public interface BinaryFormat : SerialFormat {
   public fun <T> encodeToByteArray(
       serializer: SerializationStrategy<T>,
       value: T
   ): ByteArray

   public fun <T> decodeFromByteArray(
       deserializer: DeserializationStrategy<T>,
       bytes: ByteArray
   ): T
}
```

```
public interface StringFormat : SerialFormat {
   public fun <T> encodeToString(
       serializer: SerializationStrategy<T>,
       value: T
   ): String

   public fun <T> decodeFromString(
       deserializer: DeserializationStrategy<T>,
       string: String
   ): T
}
```

```
public inline fun <reified T> StringFormat.decodeFromString(string: String): T =
    decodeFromString(serializersModule.serializer<T>(), string)
```

BUILTIN TYPES

For built-in serializers, **only primitives have serializer**, for collections it should be created explicitly

BUILTIN TYPES

PRIMITIVES

- `Boolean`
- `Byte`
- `Short`
- `Int`
- `Long`
- `Float`
- `Double`
- `Char`
- `String`
- `Enums`

For built-in serializers, **only primitives have serializer**, for collections it should be created explicitly

BUILTIN TYPES

PRIMITIVES

- Boolean
- Byte
- Short
- Int
- Long
- Float
- Double
- Char
- String
- Enums

COMPOSITES

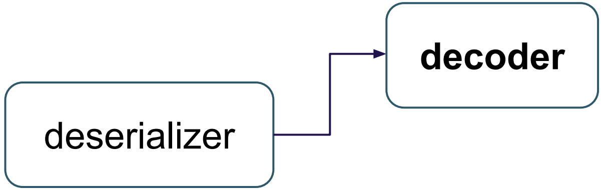
- Pairs, Triples
- Lists
- Sets
- Maps
- Objects (kotlin)

For built-in serializers, **only primitives have serializer**, for collections it should be created explicitly

HOW DOES PRIMITIVE SERIALIZER LOOKS LIKE

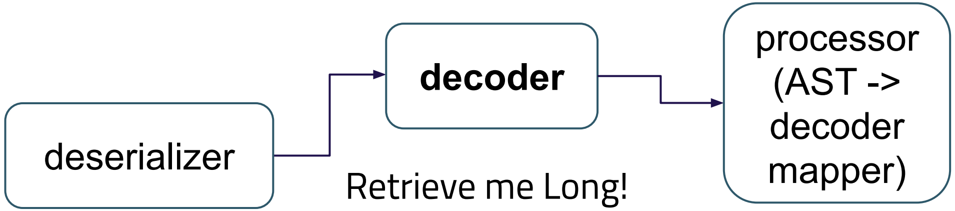
```
public fun Int.serializer(): KSerializer<Int> = IntSerializer


@PublishedApi
internal object IntSerializer : KSerializer<Int> {
   override val descriptor: SerialDescriptor = PrimitiveSerialDescriptor("kotlin.Int", PrimitiveKind.INT)
   override fun serialize(encoder: Encoder, value: Int): Unit = encoder.encodeInt(value)
   override fun deserialize(decoder: Decoder): Int = decoder.decodeInt()
}
```

DESERIALIZATION FLOW



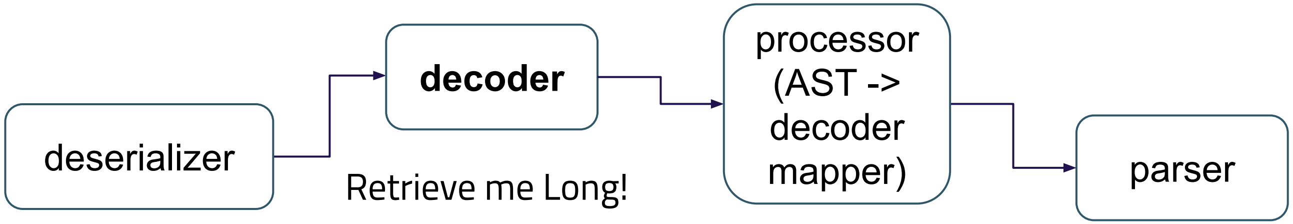
Provide me Long!

DESERIALIZATION FLOW



Provide me Long!

DESERIALIZATION FLOW



deserializer

decoder

processor (AST -> decoder mapper)

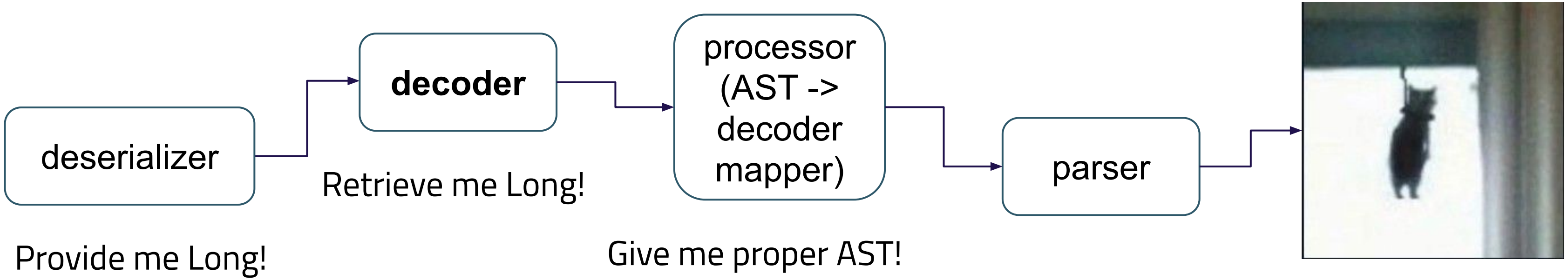
parser

Retrieve me Long!

Provide me Long!

Give me proper AST!

DESERIALIZATION FLOW



deserializer

decoder

processor (AST -> decoder mapper)

parser

Retrieve me Long!

Provide me Long!

Give me proper AST!

FORMAT-AGNOSTIC INTERFACES
DECODER / ENCODER

- Methods for **primitives**, e.g.: decodeLong(): Long / encodeInt(): Unit / etc.

FORMAT-AGNOSTIC INTERFACES DECODER / ENCODER

- Methods for **primitives**, e.g.: decodeLong(): Long / encodeInt(): Unit / etc.
- “Corner cases”: encodeNull(), decodeInline(), encodeEnum()

FORMAT-AGNOSTIC INTERFACES DECODER / ENCODER

- Methods for **primitives**, e.g.: decodeLong(): Long / encodeInt(): Unit / etc.
- “Corner cases”: encodeNull(), decodeInline(), encodeEnum()
- beginStructure(descriptor: SerialDescriptor): CompositeDecoder/Encoder

IMPORTANT ENTRY-POINT

In Decoder/Encoder:

```
public fun <T : Any?> decodeSerializableValue(
       deserializer:DeserializationStrategy<T>
    ): T = deserializer.deserialize(this)
```

IMPORTANT ENTRY-POINT

In Decoder/Encoder:

```
public fun <T : Any?> decodeSerializableValue(
        deserializer:DeserializationStrategy<T>
    ): T = deserializer.deserialize(this)
```

These methods calls `deserialize/serialize` methods and is included into the call chain

IMPORTANT ENTRY-POINT

In Decoder/Encoder:

```
public fun <T : Any?> decodeSerializableValue(
        deserializer:DeserializationStrategy<T>
    ): T = deserializer.deserialize(this)
```

These methods calls `deserialize/serialize` methods and is included into the call chain

Example for primitives:

```
decodeInt() == decodeSerializableValue(IntSerializer)
```

COMPOSITE DECODER AND ENCODER

- Utility interfaces used during the iteration process

COMPOSITE DECODER AND ENCODER

- Utility interfaces used during the iteration process

- `public fun decodeElementIndex(descriptor: SerialDescriptor): Int`

COMPOSITE DECODER AND ENCODER

- Utility interfaces used during the iteration process

- `public fun decodeElementIndex(descriptor: SerialDescriptor): Int`

-
```
loop@ while (true) {
        when (val index = decodeElementIndex(descriptor)) {
          DECODE_DONE -> break@loop
          0 -> {
                field = decodeIntElement(descriptor, index = 0)
          }
        }
}
```

Descriptor can return an index of the field by the name with *descriptor.getElementIndex(name)* , so we can always get the position

COMPOSITE DECODER AND ENCODER

- Utility interfaces used during the iteration process

- `public fun decodeElementIndex(descriptor: SerialDescriptor): Int`

-
```
loop@ while (true) {
        when (val index = decodeElementIndex(descriptor)) {
          DECODE_DONE -> break@loop
          0 -> {
                field = decodeIntElement(descriptor, index = 0)
          }
        }
}
```

Descriptor can return an index of the field by the name with *`descriptor.getElementIndex(name)`* , so we can always get the position

- CompositeDecoder.decodeElementIndex-based loop is used

SMALL PERFORMANCE TRICK

```kotlin
public fun decodeSequentially(): Boolean = false
```

SMALL PERFORMANCE TRICK

```
public fun decodeSequentially(): Boolean = false
```

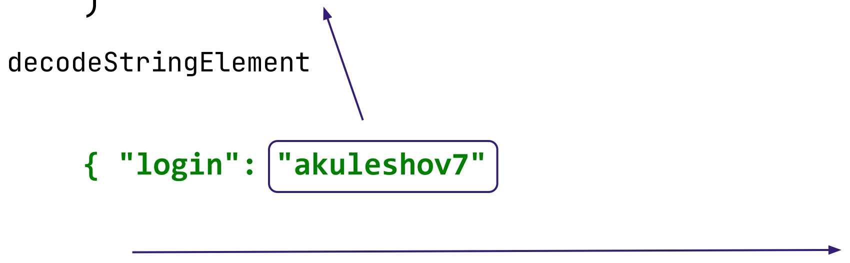
```
@Serializable
data class Credentials(
    val login: String,val password: String
)
```

```
{ "login": "akuleshov7"
```

SMALL PERFORMANCE TRICK

```
public fun decodeSequentially(): Boolean = false
```

```
@Serializable
data class Credentials(
    val login: String,val password: String
)
```



decodeStringElement

```
{ "login": "akuleshov7"
```

SMALL PERFORMANCE TRICK

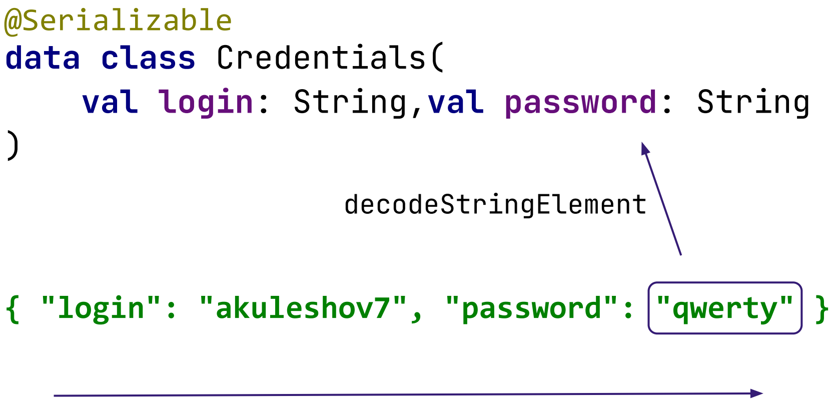
```kotlin
public fun decodeSequentially(): Boolean = false
```

```kotlin
@Serializable
data class Credentials(
    val login: String,val password: String
)
```

```json
{ "login": "akuleshov7", "password": "qwerty" }
```

SMALL PERFORMANCE TRICK

```kotlin
public fun decodeSequentially(): Boolean = false
```



```kotlin
@Serializable
data class Credentials(
    val login: String,val password: String
)
```

decodeStringElement

```json
{ "login": "akuleshov7", "password": "qwerty" }
```

ABSTRACT DECODER/ENCODER

- Predefined skeletons for simple formats

ABSTRACT DECODER/ENCODER

- Predefined skeletons for simple formats
- `abstract class AbstractDecoder : Decoder, CompositeDecoder`

ABSTRACT DECODER/ENCODER

- Predefined skeletons for simple formats
- `abstract class AbstractDecoder : Decoder, CompositeDecoder`
- `override fun decodeInt(): Int = decodeValue() as Int`

ABSTRACT DECODER/ENCODER

- Predefined skeletons for simple formats
- `abstract class AbstractDecoder : Decoder, CompositeDecoder`
- `override fun decodeInt(): Int = decodeValue() as Int`

To be implemented by library owners

TAKE A BREATH

- Let’s simplify and memorize it on the high level!

TAKE A BREATH

- Let’s simplify and memorize it on the high level!

```kotlin
@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

TAKE A BREATH

- Let’s simplify and memorize it on the high level!

```kotlin
@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

Generated **`KSerializer`** instance for `Credentials` with **`.serialize`**`()` / **`.deserialize`**`()`

TAKE A BREATH

- Let’s simplify and memorize it on the high level!

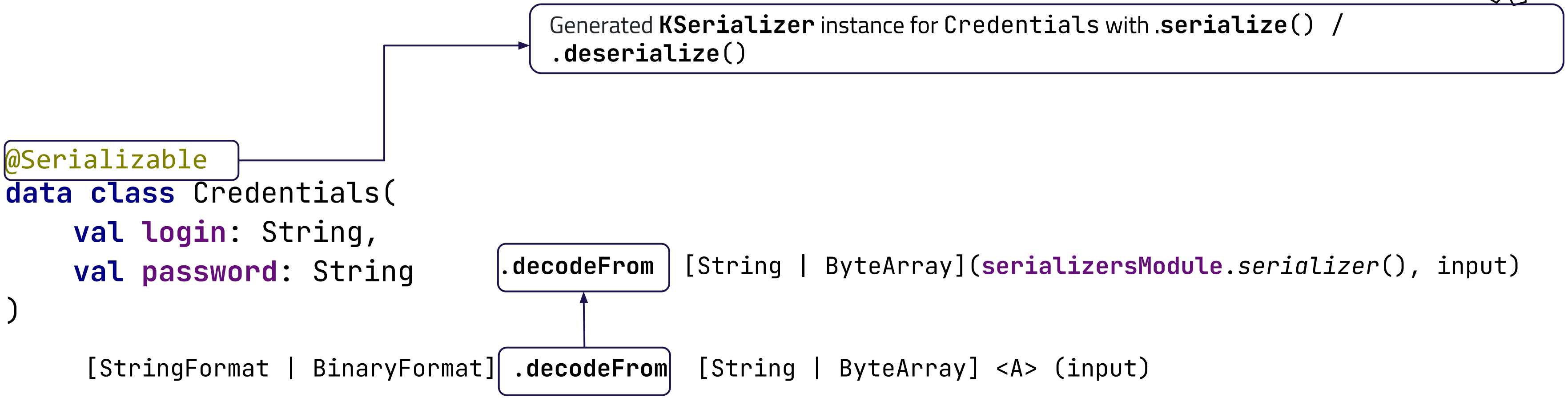
Generated **`KSerializer`** instance for `Credentials` with **`.serialize`**`()` / **`.deserialize`**`()`

```kotlin
@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

```
[StringFormat | BinaryFormat] .decodeFrom [String | ByteArray] <A> (input)
```

TAKE A BREATH

- Let’s simplify and memorize it on the high level!



Generated **KSerializer** instance for `Credentials` with `.serialize()` / `.deserialize()`

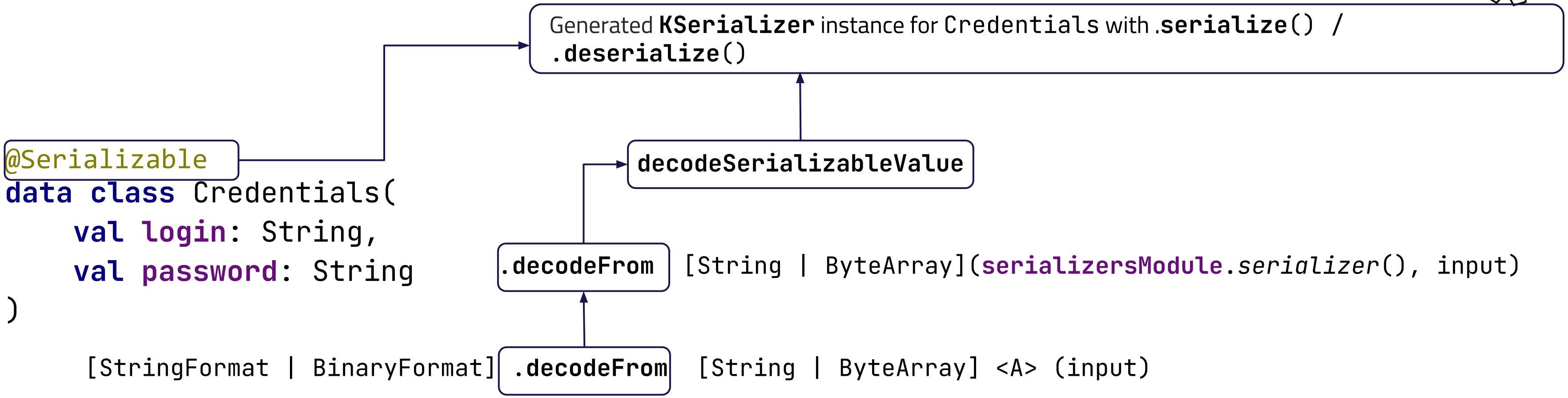
```
@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

```
.decodeFrom [String | ByteArray](serializersModule.serializer(), input)
```

```
[StringFormat | BinaryFormat] .decodeFrom [String | ByteArray] <A> (input)
```

TAKE A BREATH

- Let’s simplify and memorize it on the high level!



```kotlin
@Serializable
data class Credentials(
    val login: String,
    val password: String
)
```

Generated **KSerializer** instance for Credentials with **.serialize**() / **.deserialize**()

decodeSerializableValue

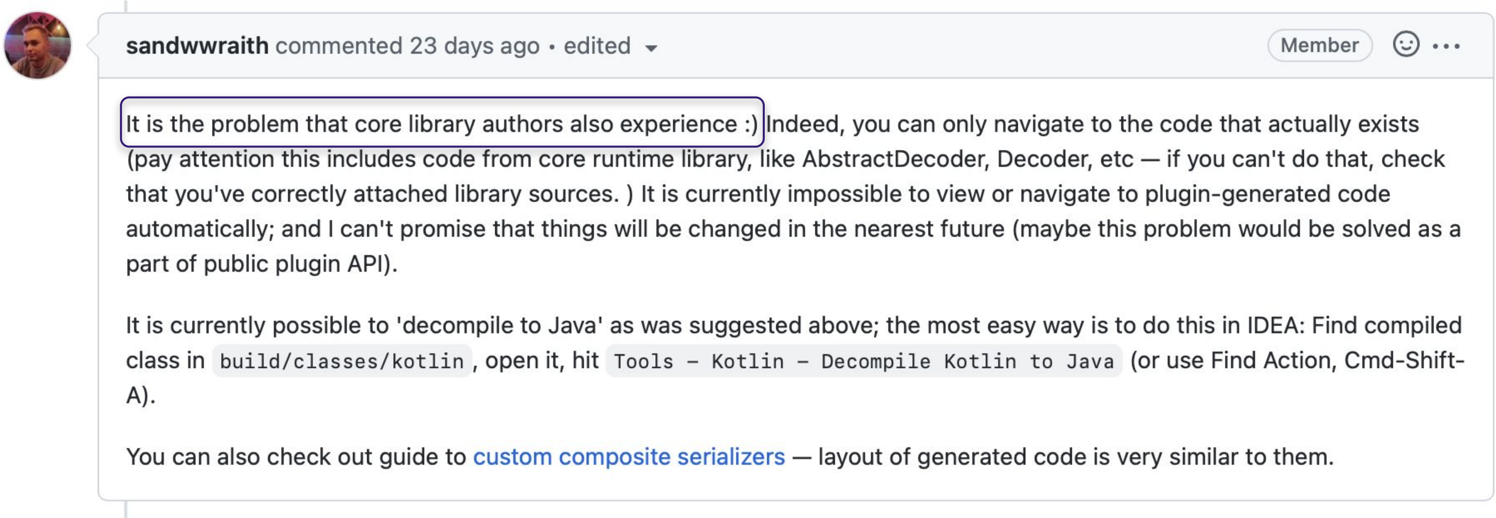
.decodeFrom [String | ByteArray](serializersModule.*serializer*(), input)

[StringFormat | BinaryFormat] **.decodeFrom** [String | ByteArray] <A> (input)

03 SERIALIZATION LIBRARIES

WE NEED TO GO DEEPER

HOW TO DEBUG?

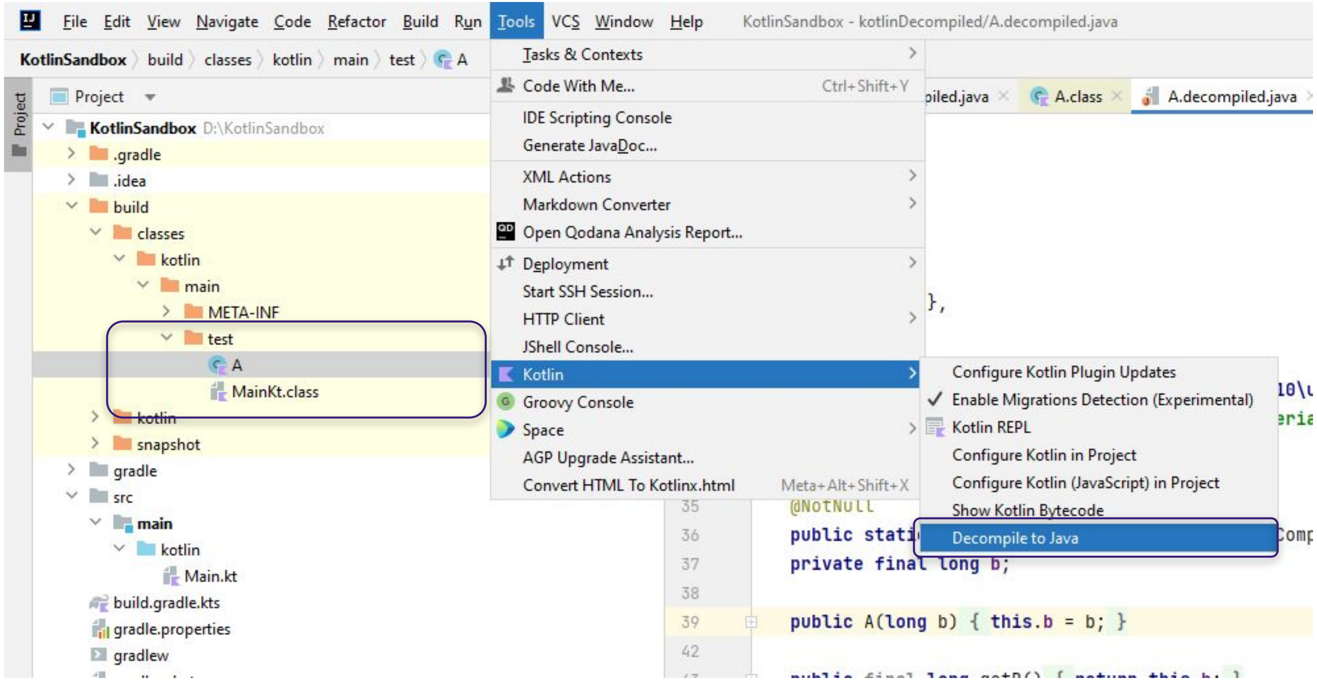


It is the problem that core library authors also experience :) Indeed, you can only navigate to the code that actually exists (pay attention this includes code from core runtime library, like AbstractDecoder, Decoder, etc — if you can't do that, check that you've correctly attached library sources.) It is currently impossible to view or navigate to plugin-generated code automatically; and I can't promise that things will be changed in the nearest future (maybe this problem would be solved as a part of public plugin API).

It is currently possible to 'decompile to Java' as was suggested above; the most easy way is to do this in IDEA: Find compiled class in `build/classes/kotlin`, open it, hit `Tools - Kotlin - Decompile Kotlin to Java` (or use Find Action, Cmd-Shift-A).

You can also check out guide to custom composite serializers — layout of generated code is very similar to them.

HOW TO DEBUG?



```
@NotNull
public stati
private final long b;

public A(long b) { this.b = b; }
```

COMPILER PLUGIN-GENERATED CODE

```
@Serializable
class A(val b: Long)
```

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
  Intrinsics.checkNotNullParameter(decoder, "decoder");
  SerialDescriptor var2 = this.getDescriptor();
  boolean var3 = true;
  int var5 = 0;
  long var6 = 0L;
  CompositeDecoder var8 = decoder.beginStructure(var2);
  if (var8.decodeSequentially()) {
     var6 = var8.decodeLongElement(var2, 0);
     var5 |= 1;
  } else {
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }

  var8.endStructure(var2);
  return new A(var5, var6, (SerializationConstructorMarker)null);
}
```

COMPILER PLUGIN-GENERATED CODE

```
@Serializable
class A(val b: Long)
```

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
  Intrinsics.checkNotNullParameter(decoder, "decoder");
  SerialDescriptor var2 = this.getDescriptor();
  boolean var3 = true;
  int var5 = 0;
  long var6 = 0L;
  CompositeDecoder var8 = decoder.beginStructure(var2);
  if (var8.decodeSequentially()) {
     var6 = var8.decodeLongElement(var2, 0);
     var5 |= 1;
  } else {
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }

  var8.endStructure(var2);
  return new A(var5, var6, (SerializationConstructorMarker)null);
}
```

COMPILER PLUGIN-GENERATED CODE

```
@Serializable
class A(val b: Long)
```

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
  Intrinsics.checkNotNullParameter(decoder, "decoder");
  SerialDescriptor var2 = this.getDescriptor();
  boolean var3 = true;
  int var5 = 0;
  long var6 = 0L;
  CompositeDecoder var8 = decoder.beginStructure(var2);
  if (var8.decodeSequentially()) {
     var6 = var8.decodeLongElement(var2, 0);
     var5 |= 1;
  } else {
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }

  var8.endStructure(var2);
  return new A(var5, var6, (SerializationConstructorMarker)null);
}
```

COMPILER PLUGIN-GENERATED CODE

```
@Serializable
class A(val b: Long)
```

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
  Intrinsics.checkNotNullParameter(decoder, "decoder");
  SerialDescriptor var2 = this.getDescriptor();
  boolean var3 = true;
  int var5 = 0;
  long var6 = 0L;
  CompositeDecoder var8 = decoder.beginStructure(var2);
  if (var8.decodeSequentially()) {
     var6 = var8.decodeLongElement(var2, 0);
     var5 |= 1;
  } else {
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }

  var8.endStructure(var2);
  return new A(var5, var6, (SerializationConstructorMarker)null);
}
```

WHAT'S INSIDE?

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
                            (...)
  CompositeDecoder var8 = decoder.beginStructure(var2);
                            (...)
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }
                             (...)
}
```

your AbstractDecoder

WHAT'S INSIDE?

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
                            (...)
  CompositeDecoder var8 = decoder.beginStructure(var2);
                            (...)
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }
                             (...)
}
```

→ `your AbstractDecoder`

→ entry point for iteration

WHAT'S INSIDE?

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
                            (...)
  CompositeDecoder var8 = decoder.beginStructure(var2);
                            (...)
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }
                             (...)
}
```

your AbstractDecoder

entry point for iteration

find the position of target field where we will write

WHAT'S INSIDE?

```
@NotNull
public A deserialize(@NotNull Decoder decoder) {
                            (...)
  CompositeDecoder var8 = decoder.beginStructure(var2);
                            (...)
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }
                             (...)
}
```

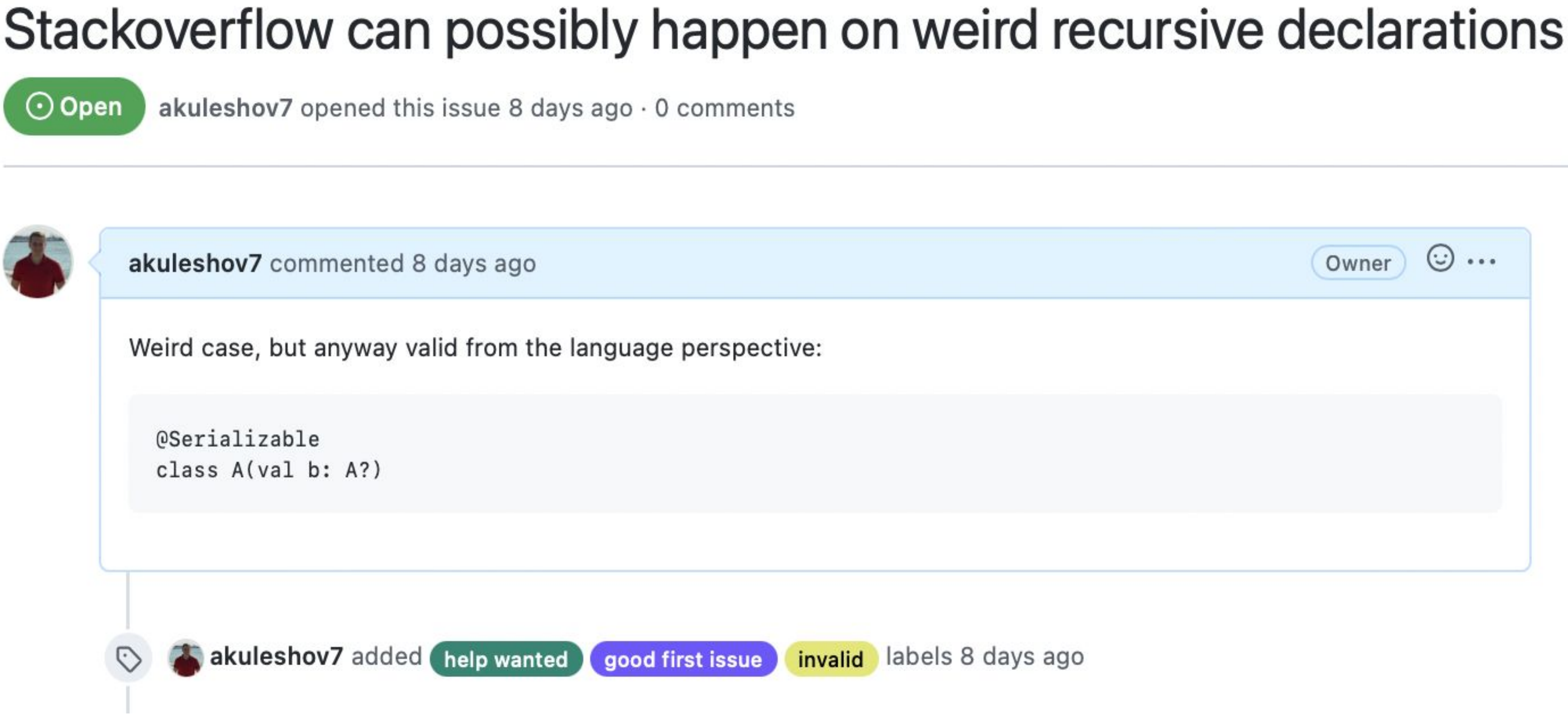
`your AbstractDecoder`

entry point for iteration

find the position of target field where we will write

DECODE_DONE

UNKNOWN_NAME (-3)



Stackoverflow can possibly happen on weird recursive declarations

⊙ Open **akuleshov7** opened this issue 8 days ago · 0 comments

akuleshov7 commented 8 days ago

Weird case, but anyway valid from the language perspective:

```
@Serializable
class A(val b: A?)
```

akuleshov7 added help wanted good first issue invalid labels 8 days ago

CALL CHAIN

How do we get there?

```kotlin
@Serializable
class A(val b: Long)

fun main() {
    Toml.decodeFromString<A>(
        """
            b = 0
        """
    )
}
```

```java
@NotNull
public A deserialize(@NotNull Decoder decoder) {
  Intrinsics.checkNotNullParameter(decoder, "decoder");
  SerialDescriptor var2 = this.getDescriptor();
  boolean var3 = true;
  int var5 = 0;
  long var6 = 0L;
  CompositeDecoder var8 = decoder.beginStructure(var2);
  if (var8.decodeSequentially()) {
     var6 = var8.decodeLongElement(var2, 0);
     var5 |= 1;
  } else {
     while(var3) {
        int var4 = var8.decodeElementIndex(var2);
        switch (var4) {
           case -1:
              var3 = false;
              break;
           case 0:
              var6 = var8.decodeLongElement(var2, 0);
              var5 |= 1;
              break;
           default:
              throw new UnknownFieldException(var4);
        }
     }
  }

  var8.endStructure(var2);
  return new A(var5, var6, (SerializationConstructorMarker)null);
}
```

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")
```

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
     deserializer: DeserializationStrategy<T>,
     string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}
```

```
class Toml : StringFormat()
```

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
    deserializer: DeserializationStrategy<T>,
    string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}
```

Inject your parser

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
    deserializer: DeserializationStrategy<T>,
    string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}
```

```
class TomlMainDecoder : AbstractDecoder
```

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
    deserializer: DeserializationStrategy<T>,
    string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}

fun <T> decode(
    deserializer: DeserializationStrategy<T>,
    rootNode: TomlNode
): T {
   val decoder = TomlMainDecoder(rootNode)
   return decoder.decodeSerializableValue(deserializer)
}
```

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
    deserializer: DeserializationStrategy<T>,
    string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}

fun <T> decode(
    deserializer: DeserializationStrategy<T>,
    rootNode: TomlNode
): T {
   val decoder = TomlMainDecoder(rootNode)
   return decoder.decodeSerializableValue(deserializer)
}
```

Inject your Decoder

SIMPLIFIED CALL CHAIN EXAMPLE

```
Toml.decodeFromString<A>("")

override fun <T> decodeFromString(
    deserializer: DeserializationStrategy<T>,
    string: String
): T {
   val parsedToml = tomlParser.parseString(string)
   return TomlMainDecoder.decode(deserializer, parsedToml)
}

fun <T> decode(
    deserializer: DeserializationStrategy<T>,
    rootNode: TomlNode
): T {
   val decoder = TomlMainDecoder(rootNode)
   return decoder.decodeSerializableValue(deserializer)
}
```

AST

- Native ⇨ cannot reuse existing JVM parsers

AST

- Native ⇨ cannot reuse existing JVM parsers
- Start from the creation of a proper Parser to AST

AST

- Native ⇨ cannot reuse existing JVM parsers
- Start from the creation of a proper Parser to AST
- Main goal is to create an IR that will be easy to iterate

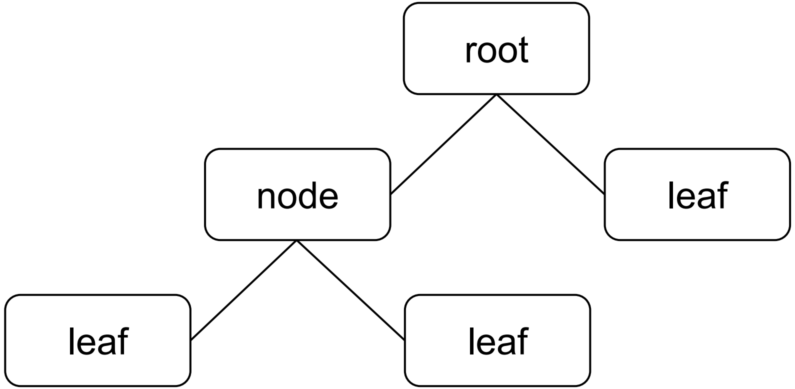
AST

- Native ⇨ cannot reuse existing JVM parsers
- Start from the creation of a proper Parser to AST
- Main goal is to create an IR that will be easy to iterate

[A] b = 1	{ "A": { "b": 1 } }	A: b: 1
Parser	Lexer	Parser
KTOML	JSON	KAML

MAPPING FROM AST TO OBJECT

- Custom decoder/encoder have an iteration index and “begin structure” with root node



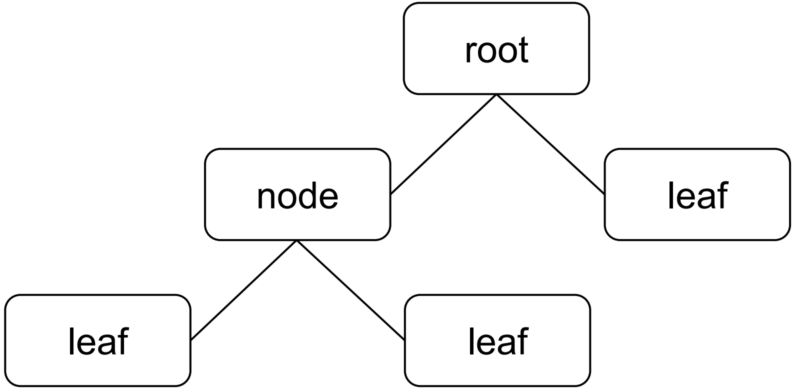
```
a = 1

[table]
   b = 1
   c = 1
```

```kotlin
@Serializable
data class Toml(val a: Long, val table: Table)


@Serializable
data class Table(val b: Long, val c: Long)
```

MAPPING FROM AST TO OBJECT

- Custom decoder/encoder have an iteration index and “begin structure” with root node

- In while-loop they return indexes of a field by name and increment index



```toml
a = 1

[table]
   b = 1
   c = 1
```

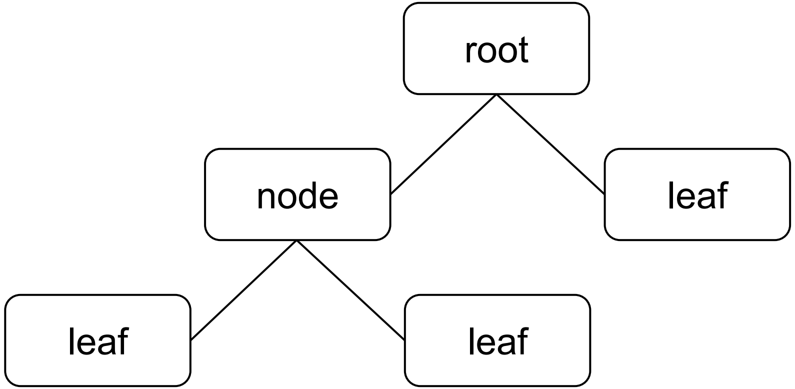
```kotlin
@Serializable
data class Toml(val a: Long, val table: Table)


@Serializable
data class Table(val b: Long, val c: Long)
```

MAPPING FROM AST TO OBJECT

- Custom decoder/encoder have an iteration index and “begin structure” with root node

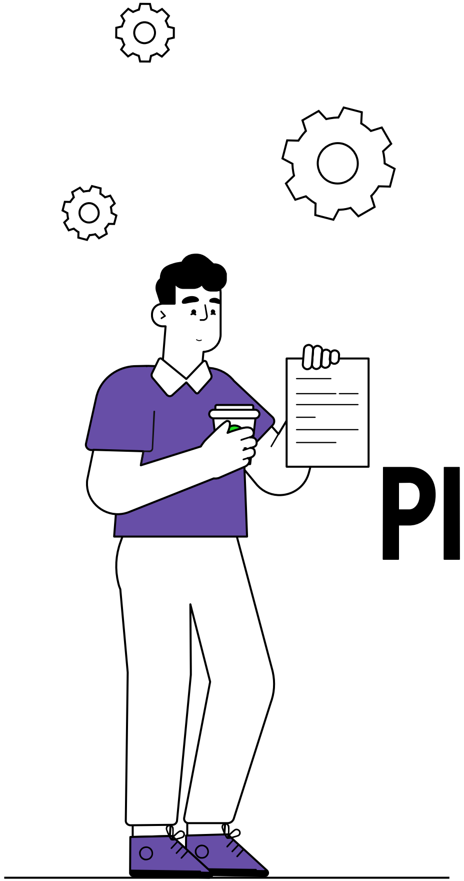
- In while-loop they return indexes of a field by name and increment index

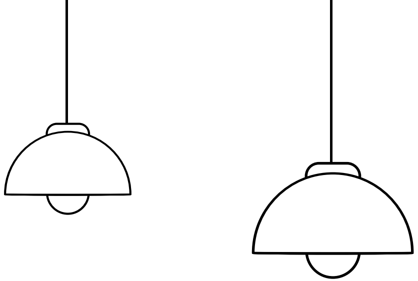
- Using the index proper field is decoded (compiler plugin knows the type and calls proper decoder)



```toml
a = 1

[table]
   b = 1
   c = 1
```

```kotlin
@Serializable
data class Toml(val a: Long, val table: Table)


@Serializable
data class Table(val b: Long, val c: Long)
```

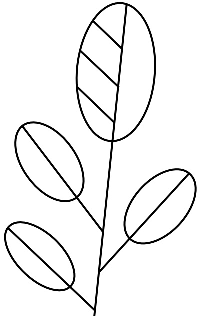




04

PITFALLS AND CONCLUSION

Smart people learn from mistakes of others



MISTAKES I MADE

- Did not read carefully `kotlinx.serialization` guide

MISTAKES I MADE

- Did not read carefully `kotlinx.serialization` guide
- Started from reading the library instead of checking **decompiled sources**

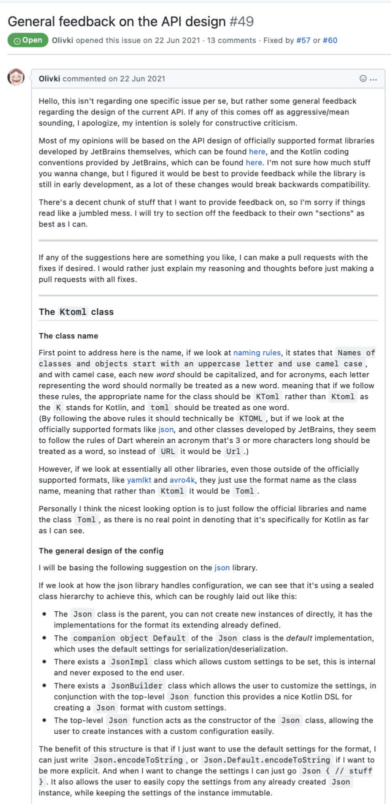
MISTAKES I MADE

- Did not read carefully `kotlinx.serialization` guide
- Started from reading the library instead of checking **decompiled sources**
- AST was good for decoding, but not perfect for encoding

MISTAKES I MADE

- Did not read carefully `kotlinx.serialization` guide
- Started from reading the library instead of checking **decompiled sources**
- AST was good for decoding, but not perfect for encoding
- Forgot about **corner cases**: nullability/inline classes/etc

ADVICES



General feedback on the API design #49

Open Olivki opened this issue on 22 Jun 2021 · 13 comments · Fixed by #57 or #60

Olivki commented on 22 Jun 2021

Hello, this isn't regarding one specific issue per se, but rather some general feedback regarding the design of the current API. If any of this comes off as aggressive/mean sounding, I apologize, my intention is solely for constructive criticism.

Most of my opinions will be based on the API design of officially supported format libraries developed by JetBrains themselves, which can be found here, and the Kotlin coding conventions provided by JetBrains, which can be found here. I'm not sure how much stuff you wanna change, but I figured it would be best to provide feedback while the library is still in early development, as a lot of these changes would break backwards compatibility.

There's a decent chunk of stuff that I want to provide feedback on, so I'm sorry if things read like a jumbled mess. I will try to section off the feedback to their own "sections" as best as I can.

If any of the suggestions here are something you like, I can make a pull requests with the fixes if desired. I would rather just explain my reasoning and thoughts before just making a pull requests with all fixes.

The `Ktoml` class

The class name

First point to address here is the name, if we look at naming rules, it states that `Names of classes and objects start with an uppercase letter and use camel case`, and with camel case, each new *word* should be capitalized, and for acronyms, each letter representing the word should normally be treated as a new word. meaning that if we follow these rules, the appropriate name for the class should be `KToml` rather than `Ktoml` as the `K` stands for Kotlin, and `toml` should be treated as one word.
(By following the above rules it should technically be `KTOML`, but if we look at the officially supported formats like json, and other classes developed by JetBrains, they seem to follow the rules of Dart wherein an acronym that's 3 or more characters long should be treated as a word, so instead of `URL` it would be `Url`.)

However, if we look at essentially all other libraries, even those outside of the officially supported formats, like yamlkt and avro4k, they just use the format name as the class name, meaning that rather than `Ktoml` it would be `Toml`.

Personally I think the nicest looking option is to just follow the official libraries and name the class `Toml`, as there is no real point in denoting that it's specifically for Kotlin as far as I can see.

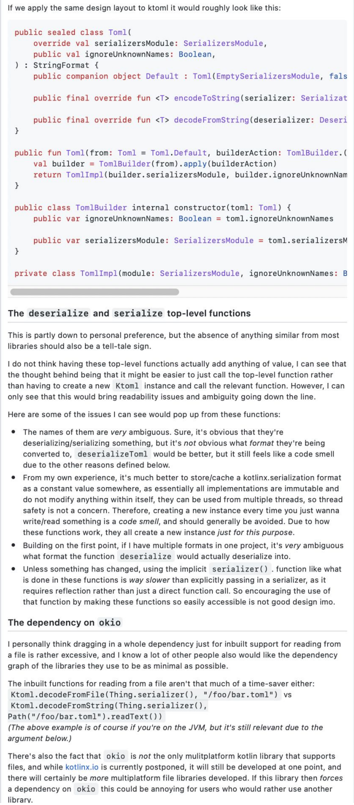
The general design of the config

I will be basing the following suggestion on the json library.

If we look at how the json library handles configuration, we can see that it's using a sealed class hierarchy to achieve this, which can be roughly laid out like this:

- The `Json` class is the parent, you can not create new instances of directly, it has the implementations for the format its extending already defined.
- The `companion object Default` of the `Json` class is the *default* implementation, which uses the default settings for serialization/deserialization.
- There exists a `JsonImpl` class which allows custom settings to be set, this is internal and never exposed to the end user.
- There exists a `JsonBuilder` class which allows the user to customize the settings, in conjunction with the top-level `Json` function this provides a nice Kotlin DSL for creating a `Json` format with custom settings.
- The top-level `Json` function acts as the constructor of the `Json` class, allowing the user to create instances with a custom configuration easily.

The benefit of this structure is that if I just want to use the default settings for the format, I can just write `Json.encodeToString`, or `Json.Default.encodeToString` if I want to be more explicit. And when I want to change the settings I can just go `Json { // stuff }`. It also allows the user to easily copy the settings from any already created `Json` instance, while keeping the settings of the instance immutable.



If we apply the same design layout to ktoml it would roughly look like this:

```
public sealed class Toml(
    override val serializersModule: SerializersModule,
    public val ignoreUnknownNames: Boolean,
) : StringFormat {
    public companion object Default : Toml(EmptySerializersModule, fals

    public final override fun <T> encodeToString(serializer: Serializat

    public final override fun <T> decodeFromString(deserializer: Deseri
}

public fun Toml(from: Toml = Toml.Default, builderAction: TomlBuilder.(
    val builder = TomlBuilder(from).apply(builderAction)
    return TomlImpl(builder.serializersModule, builder.ignoreUnknownNam
}

public class TomlBuilder internal constructor(toml: Toml) {
    public var ignoreUnknownNames: Boolean = toml.ignoreUnknownNames

    public var serializersModule: SerializersModule = toml.serializersM
}

private class TomlImpl(module: SerializersModule, ignoreUnknownNames: B
```

The `deserialize` and `serialize` top-level functions

This is partly down to personal preference, but the absence of anything similar from most libraries should also be a tell-tale sign.

I do not think having these top-level functions actually add anything of value, I can see that the thought behind being that it might be easier to just call the top-level function rather than having to create a new `Ktoml` instance and call the relevant function. However, I can only see that this would bring readability issues and ambiguity going down the line.

Here are some of the issues I can see would pop up from these functions:

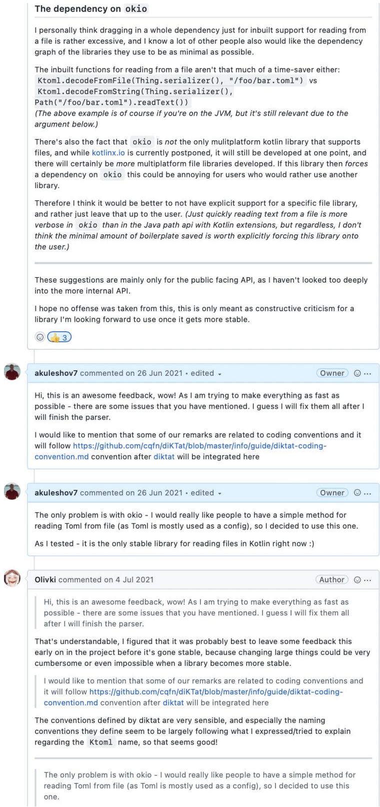
- The names of them are *very* ambiguous. Sure, it's obvious that they're deserializing/serializing something, but it's *not* obvious what *format* they're being converted to, `deserializeToml` would be better, but it still feels like a code smell due to the other reasons defined below.
- From my own experience, it's much better to store/cache a kotlinx.serialization format as a constant value somewhere, as essentially all implementations are immutable and do not modify anything within itself, they can be used from multiple threads, so thread safety is not a concern. Therefore, creating a new instance every time you just wanna write/read something is a *code smell*, and should generally be avoided. Due to how these functions work, they all create a new instance *just for this purpose*.
- Building on the first point, if I have multiple formats in one project, it's *very* ambiguous what format the function `deserialize` would actually deserialize into.
- Unless something has changed, using the implicit `serializer()`. function like what is done in these functions is *way slower* than explicitly passing in a serializer, as it requires reflection rather than just a direct function call. So encouraging the use of that function by making these functions so easily accessible is not good design imo.

The dependency on `okio`

I personally think dragging in a whole dependency just for inbuilt support for reading from a file is rather excessive, and I know a lot of other people also would like the dependency graph of the libraries they use to be as minimal as possible.

The inbuilt functions for reading from a file aren't that much of a time-saver either:
`Ktoml.decodeFromFile(Thing.serializer(), "/foo/bar.toml")` vs
`Ktoml.decodeFromString(Thing.serializer(), Path("/foo/bar.toml").readText())`
(The above example is of course if you're on the JVM, but it's still relevant due to the argument below.)

There's also the fact that `okio` is *not* the only mulitplatform kotlin library that supports files, and while kotlinx.io is currently postponed, it will still be developed at one point, and there will certainly be *more* multiplatform file libraries developed. If this library then *forces* a dependency on `okio` this could be annoying for users who would rather use another library.



The dependency on `okio`

I personally think dragging in a whole dependency just for inbuilt support for reading from a file is rather excessive, and I know a lot of other people also would like the dependency graph of the libraries they use to be as minimal as possible.

The inbuilt functions for reading from a file aren't that much of a time-saver either:
`Ktoml.decodeFromFile(Thing.serializer(), "/foo/bar.toml")` vs
`Ktoml.decodeFromString(Thing.serializer(), Path("/foo/bar.toml").readText())`
(The above example is of course if you're on the JVM, but it's still relevant due to the argument below.)

There's also the fact that `okio` is *not* the only mulitplatform kotlin library that supports files, and while kotlinx.io is currently postponed, it will still be developed at one point, and there will certainly be *more* multiplatform file libraries developed. If this library then *forces* a dependency on `okio` this could be annoying for users who would rather use another library.

Therefore I think it would be better to not have explicit support for a specific file library, and rather just leave that up to the user. *(Just quickly reading text from a file is more verbose in `okio` than in the Java path api with Kotlin extensions, but regardless, I don't think the minimal amount of boilerplate saved is worth explicitly forcing this library onto the user.)*

These suggestions are mainly only for the public facing API, as I haven't looked too deeply into the more internal API.

I hope no offense was taken from this, this is only meant as constructive criticism for a library I'm looking forward to use once it gets more stable.

👍 3

akuleshov7 commented on 26 Jun 2021 • edited Owner

Hi, this is an awesome feedback, wow! As I am trying to make everything as fast as possible - there are some issues that you have mentioned. I guess I will fix them all after I will finish the parser.

I would like to mention that some of our remarks are related to coding conventions and it will follow https://github.com/cqfn/diKTat/blob/master/info/guide/diktat-coding-convention.md convention after diktat will be integrated here

akuleshov7 commented on 26 Jun 2021 • edited Owner

The only problem is with okio - I would really like people to have a simple method for reading Toml from file (as Toml is mostly used as a config), so I decided to use this one.

As I tested - it is the only stable library for reading files in Kotlin right now :)

Olivki commented on 4 Jul 2021 Author

> Hi, this is an awesome feedback, wow! As I am trying to make everything as fast as possible - there are some issues that you have mentioned. I guess I will fix them all after I will finish the parser.

That's understandable, I figured that it was probably best to leave some feedback this early on in the project before it's gone stable, because changing large things could be very cumbersome or even impossible when a library becomes more stable.

> I would like to mention that some of our remarks are related to coding conventions and it will follow https://github.com/cqfn/diKTat/blob/master/info/guide/diktat-coding-convention.md convention after diktat will be integrated here

The conventions defined by diktat are very sensible, and especially the naming conventions they define seem to be largely following what I expressed/tried to explain regarding the `Ktoml` name, so that seems good!

> The only problem is with okio - I would really like people to have a simple method for reading Toml from file (as Toml is mostly used as a config), so I decided to use this one.

CONCLUSION

- Migrate to kotlinx.serialization

CONCLUSION

- Migrate to kotlinx.serialization
- Participate in open-source development

CONCLUSION

- Migrate to kotlinx.serialization
- Participate in open-source development
- Give stars on the GitHub to projects (it really helps to attract contributors!)
- akuleshov7/ktoml
- charleskorn/kaml
- kotlin/kotlinx.serialization

[KT]oml

powered by kotlinx.serialization

THANK YOU FOR LISTENING!

Special thanks to:

@NightEule5,
@bishiboosh,
@Peanuuutz,
@petertrr,
@nulls,
@Olivki
@icemachined,
@unix-junkie