Kotlin in Practice

Philipp Hauer, Fabian Sudau
Spreadshirt

JUG Saxony Camp 2017

Spreadshirt

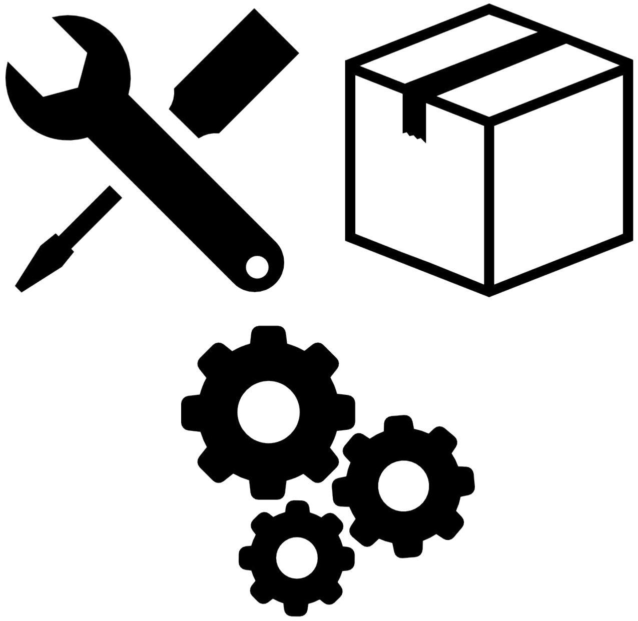
Hands Up!

Introduction

Introduction: Java Ecosystem and Language

Powerful Ecosystem

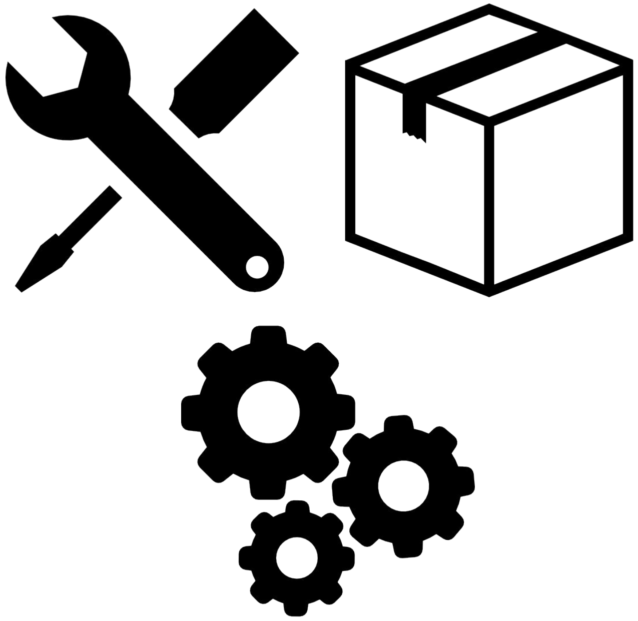
Poor Language



Introduction: Java Ecosystem and Language

Powerful Ecosystem

Powerful Language



Ecosystem vs. Language

Introduction: Kotlin

- By JetBrains
- 2011
- Open-Source
- Characteristics:
  - Statically Typed
  - Object-Oriented
  - Many Functional Concepts
- Compiled to Java Bytecode
- Runs on the Java Virtual Machine (JVM)
- “Pragmatic”

Language Features

Language Features: Immutability

Support for immutability *built-in*

```
// mutable state - use rarely
var mutableList = mutableListOf(1, 2, 3)
mutableList.add(4)

// 'immutable' state - generally recommended
val readonlyList = listOf(1, 2, 3)
~~readonlyList.add(4)~~
~~readonlyList = listOf(1, 2, 3, 4)~~
```

→ Immutability feels defaultish, encourages better design
→ Still 100% Java compatible due to interface-based approach

Language Features: Data Classes

Concise Data Containers

```
// implies sensible defaults for equals(...), hashCode(), toString()
data class Person(val firstName: String, var age: Int,
    val title: Title = Title.UNKNOWN)

val jon = Person(firstName = "Jon", age = 18, title = Title.MR)
val defaultJon = Person(firstName = "Jon", age = 18)
val jack = jon.copy(firstName = "Jack")

println(jack.firstName)  // prints 'Jack'
jack.age = 19 // assignment of var
```
~~jack.firstName = "Joseph"~~ `// illegal, field is immutable`

→ Plain Java equivalent requires 91 LOC!

Language Features: Null Safety

“Billion dollar mistake” in Java: Kotlin proposes an alternative

```
var name: String = "Jon"
println(name.length)
~~name = null~~ // does not compile


var nullableName: String? = "Jon"
~~println(nullableName.length)~~ // does not compile
println(nullableName?.length) // 3 or null
println(nullableName?.length ?: 0) // 3 or 0
if (nullableName != null) {
   println(nullableName.length) // compiler infers safety
}
nullableName = null // compiles
```

Language Features: Type Inference

Local type inference (compromise)

```
fun doubleLength(str: String): Int {
    val stringLength = str.length
    // == val stringLength: Int = str.length
    return stringLength * 2
}
```

Single Expression Functions

```
fun doubleLength(str: String) = str.length * 2
// return type inferred
```

Inference on immediate assignment

```
class Foo{
    val bar = "baz"  // == val bar: String = "baz"
}
```

Language Features: Devs just wanna have *fun*

Top-Level Functions

```
// Hello World.kt
fun main(args: Array<String>) {
   println("Hello World")
}
```

```
// HelloWorld.java
public class HelloWorld {
   public static void main(String[] args) {
       System.out.println( "Hello World");
   }
}
```

Extension functions

```
// definition
fun String.wrap(wrapWith: String) = wrapWith + this + wrapWith
// usage
val wrapped = "hello".wrap("*")
// as opposed to:
val wrapped = StringUtils.wrap("hello", "*")
```

Language Features: Devs just wanna have *fun*

Proper function types

```
fun foo(provideString: () -> String,
          consumeString: (String) -> Int): Int {
     return consumeString(provideString())
}
```

Concise Lambdas

```
fun safe(code: () -> Unit) {
    try {
         code()
    } catch(e: Exception) {
         log.ERROR("Exception t_t", e)
    }
}
// usage
safe {
    println("Hello World")
}
```

Language Features: Smart Pattern Matching

When - A better Switch Case

```
// myInt: Int
val x = when (myInt) {
   0 -> "zero"
   1, 2 -> "one or two"
   in 3..10 -> "small number"
   in 11..Int.MAX_VALUE -> "large number"
   else -> "negative number"
}

enum class Mode { ON, OFF, IDLE }

fun messageForMode(mode: Mode) = when (mode) {
   Mode.ON -> "Turning on"
   Mode.OFF -> "Turning off"
}
// error: 'when' expression is not exhaustive
```

Kotlin at Spreadshirt: Evaluation

Welcome to the Enterprise: No Changes without Careful Consideration

Management Matrix ™

Pros	**Cons**
Reuse Java ecosystem (big deal!)	Niche
Reuse beloved Java frameworks & libs	Know-how required
Easy to learn: Conservative language concepts	Risk: Unexpected complications?
Less error prone: NPEs and mutable state easier to avoid	Future of language highly depends on JetBrains
Conciseness: Improved maintainability, less developer frustration	.. but we already have *Scala*

Giving it a Try

Start <u>small</u>:

- Migrate a system test suite from Java to Kotlin
- Carve out “misplaced” functionality from a Java monolith to a Kotlin based microservice
- Re-evaluate based on this experience
    - Advantages significant enough to use Kotlin for the next project?

Kotlin at Spreadshirt: Usage

Kotlin Usage at Spreadshirt

5 new services purely written in Kotlin

1 Java service enriched with Kotlin

Kotlin Usage at Spreadshirt

Kotlin in conjunction with popular Java frameworks and tools

spring

Jersey

Jackson

vaadin}>

JUnit

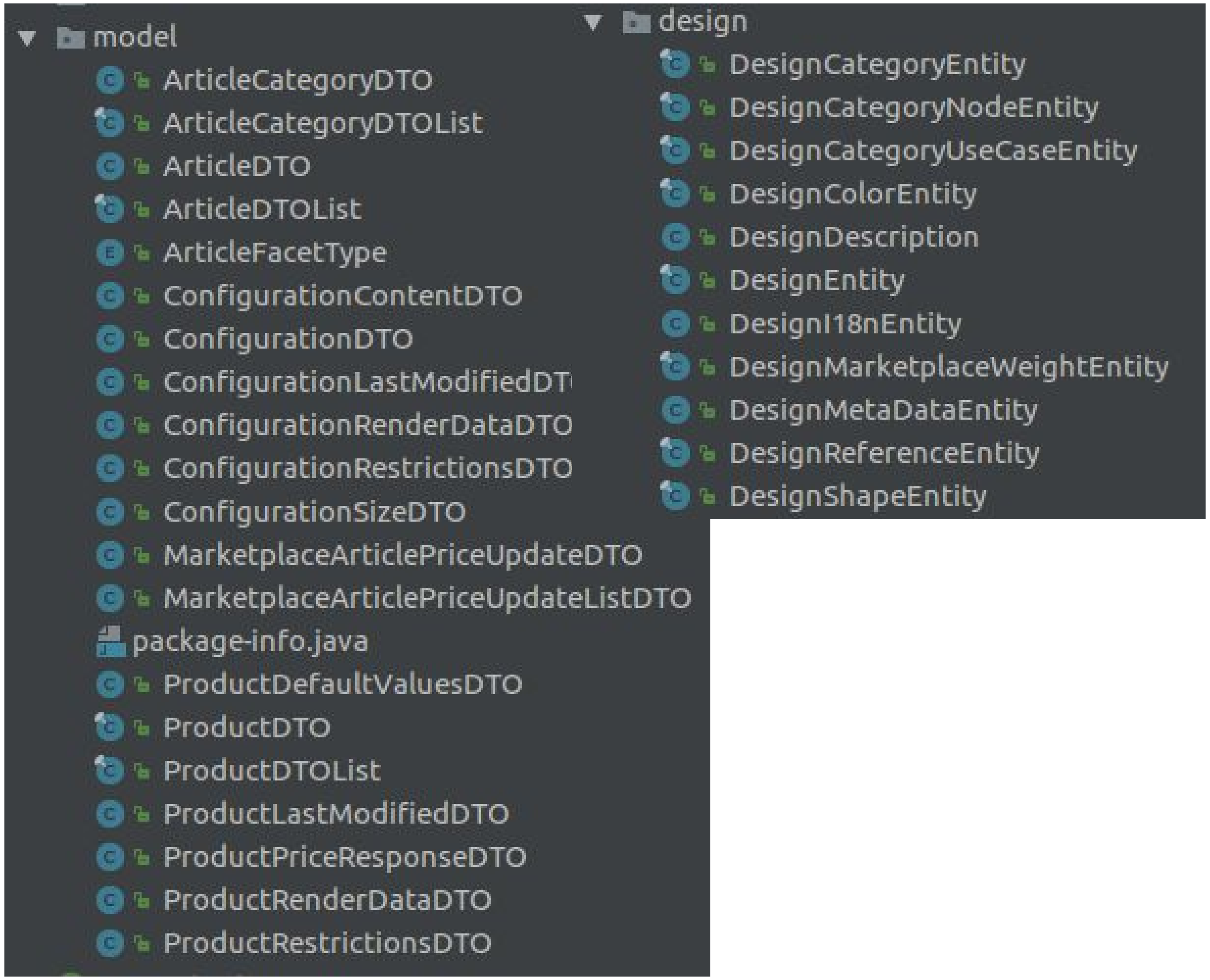
maven

REST-assured

Taking Advantage of Kotlin in Practice

Putting Classes Together

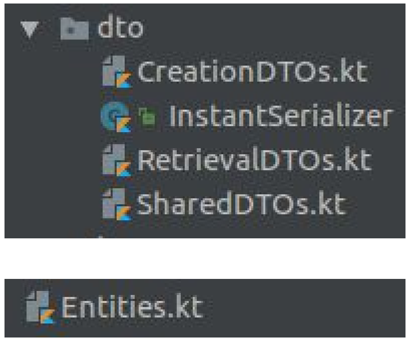
Java



model
- ArticleCategoryDTO
- ArticleCategoryDTOList
- ArticleDTO
- ArticleDTOList
- ArticleFacetType
- ConfigurationContentDTO
- ConfigurationDTO
- ConfigurationLastModifiedDT
- ConfigurationRenderDataDTO
- ConfigurationRestrictionsDTO
- ConfigurationSizeDTO
- MarketplaceArticlePriceUpdateDTO
- MarketplaceArticlePriceUpdateListDTO
- package-info.java
- ProductDefaultValuesDTO
- ProductDTO
- ProductDTOList
- ProductLastModifiedDTO
- ProductPriceResponseDTO
- ProductRenderDataDTO
- ProductRestrictionsDTO

design
- DesignCategoryEntity
- DesignCategoryNodeEntity
- DesignCategoryUseCaseEntity
- DesignColorEntity
- DesignDescription
- DesignEntity
- DesignI18nEntity
- DesignMarketplaceWeightEntity
- DesignMetaDataEntity
- DesignReferenceEntity
- DesignShapeEntity

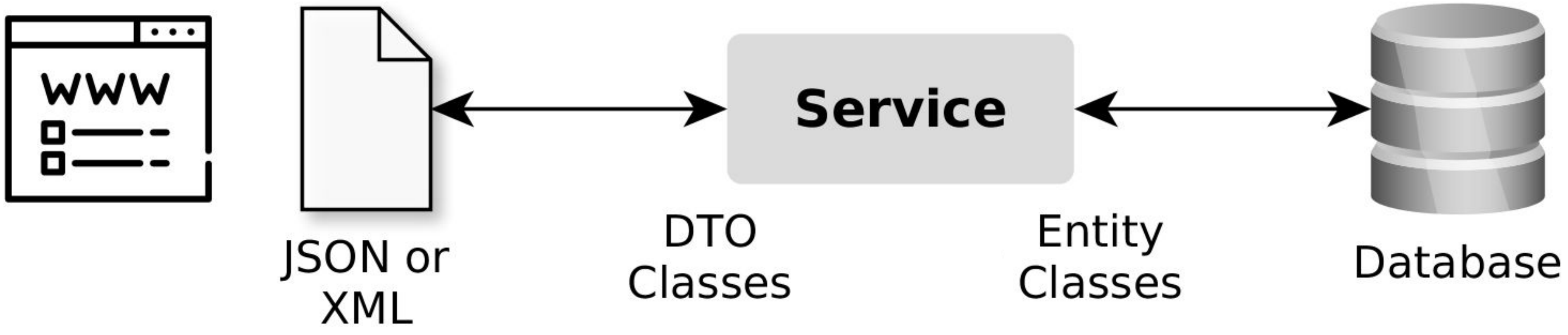
Kotlin



dto
- CreationDTOs.kt
- InstantSerializer
- RetrievalDTOs.kt
- SharedDTOs.kt

Entities.kt

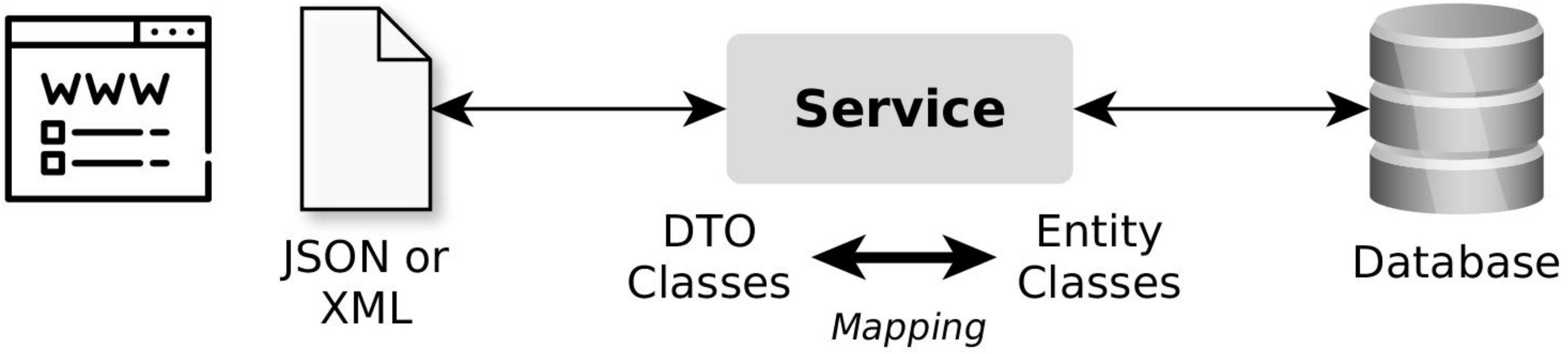
Concise Mapping between Model Classes



```
data class SnippetDTO(
    val code: String,
    val author: String,
    val date: Instant
)
```

```
data class SnippetEntity(
    val code: String,
    val author: AuthorEntity,
    val date: Instant
)
data class AuthorEntity(
    val firstName: String,
    val lastName: String
)
```

Concise Mapping between Model Classes



```
fun mapToDTO(entity: SnippetEntity) = SnippetDTO(
        code = entity.code,
        date = entity.date,
        author = "${entity.author.firstName}
${entity.author.lastName}"
)
```

Value Objects

```
//without value object:
fun send(target: String){}

//expressive, readable, safe
fun send(target: EmailAddress){}

//with value object:
data class EmailAddress(val value: String)
```

Vaadin: Structuring UI Definition with `apply()`

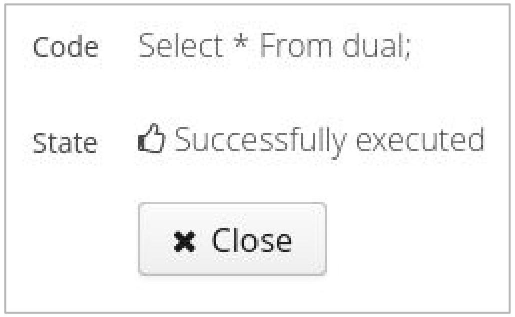
```
//Java:
Table myTable = new Table("MyTable", container);
myTable.setSizeFull();
myTable.setColumnHeader("code", "Code");
myTable.setColumnHeader("date", "Date");
myTable.addGeneratedColumn("code", ShortValueColumnGenerator);
myTable.setConverter("date", StringToInstantConverter);
```

Vaadin: Structuring UI Definition with `apply()`

```
//Kotlin:
val myTable = Table("MyTable", container).apply {
    setSizeFull()
    setColumnHeader("code", "Code")
    setColumnHeader("date", "Date")
    addGeneratedColumn("code", ShortenedValueColumnGenerator)
    setConverter("date", StringToInstantConverter)
}
```

Vaadin: Structuring UI Definition with `apply()`

```
val layout = FormLayout().apply {
   setMargin(true)
   isSpacing = true
   val codeLabel = Label().apply {
       caption = "Code"
       value = "Select * From dual;"
   }
   val stateLabel = Label().apply {
       caption = "State"
       value = "${icon} Successfully executed"
   }
   val closeButton = Button("Close").apply {
       addClickListener { close() }
   }
   addComponents(codeLabel, stateLabel, closeButton)
}
```


Code Select * From dual;

State Successfully executed

Vaadin: Extension Functions to Add UI Logic

```
enum class SnippetState {EXECUTED, NOT_EXECUTED}


fun SnippetState.toIcon() = when (this){
    SnippetState.EXECUTED -> FontAwesome.THUMBS_O_UP
    SnippetState.NOT_EXECUTED -> FontAwesome.THUMBS_O_DOWN
}

//usage:
val icon = state.toIcon()
```

Popular Java Idioms and Patterns are Built-in

Java Idiom or Pattern	**Idiomatic Solution in Kotlin**
Getter, Setter, Backing Field	Properties
Static Utility Class	Top-Level (extension) functions
Immutability, Value Objects	`data class` with Immutable Properties, `copy()`
Fluent Setter (Wither)	Named and Default Arguments, `apply()`
Method Chaining	Default Arguments
Singleton	`object`
Delegation	Delegated Properties `by`
Lazy Initialization	Delegated Properties `by: lazy()`
Observer	Delegated Properties `by: observable()`

Drawbacks and Pitfalls

Trouble with Object Mapping and XML

```
data class SnippetDTO(val code: String, val author: String)

val snippet = Snippet()
```

- JAXB (XML) requires parameterless constructor ↯ Kotlin
- Jackson (JSON) supports parameterless constructors.
  - But poor and buggy XML support
- Can’t find data class working for all use cases:
  - Jackson
  - XML and JSON
  - Serialization and Deserialization
  - Nasty XML structure
- Solution: Different data classes for Serialization and Deserialization

Final by Default

```
class CustomerService {
    fun findCustomer(id: Int){
        //...
    }
}
```

Can’t be extended by subclasses!

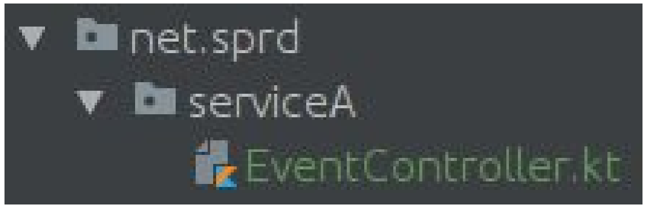
- Some frameworks rely on extension of classes
  - Spring
  - Mockito
- Solutions:
  - `Open` classes and methods explicitly
  - Open-all-plugin for Kotlin compiler

```
open class CustomerService {
    open fun findCustomer(id: Int){
        //...
    }
}
```

Works!

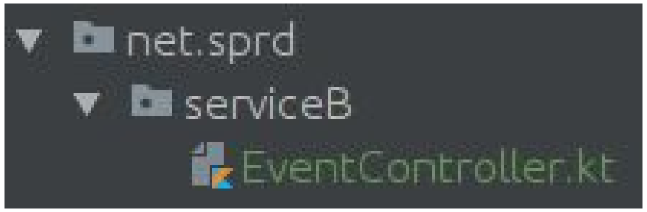
Pitfalls with Auto-Discovery via Reflection

- Java: file path = package



```
package net.sprd.ServiceA;
```

- Kotlin: no such requirement
- Copy and Paste class `EventController` to service<u>B</u>



```
package net.sprd.ServiceA
```

EventController not found!

Conclusion

Conclusion

- Programming for the JVM never felt so satisfying
- We saved ~75% of the code
- Several best practices were more prevalent in Kotlin
- Spring ecosystem worked well
- XML data binding was a major pain point and cost us days

→ Use of Kotlin will continue at Spreadshirt

Questions?