Kotlin
In Practice

@philipp_hauer
Spreadshirt

JavaLand, 13.03.18

Spreadshirt

Hands Up!

Kotlin Features and Usage in Practice

Data Classes

Immutability made easy

```
data class DesignData(
        val fileName: String,
        val uploaderId: Int,
        val width: Int = 0,
        val height: Int = 0
)
```

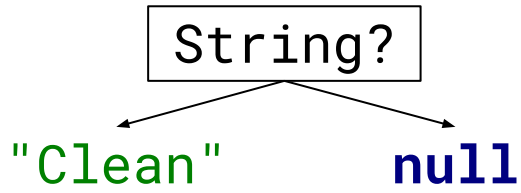
- Constructor (assign args to fields)
- Getter
- toString()
- hashCode(), equals()
- copy()
- final
- Default Arguments (no chaining)

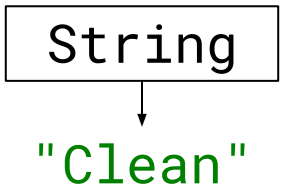
```
val design = DesignData(fileName = "cat.jpg", uploaderId = 2)

val fileName = design.fileName
design.fileName = "dog.jpg"

val design2 = design.copy(fileName = "dog.jpg")
```

Null-Safety and Means for Null Handling






```
val value: String = "Clean Code"
val value: String = null

val nullableValue: String? = "Clean Code"
val nullableValue: String? = null
val v: String = nullableValue
```

smart-cast!

```
val v: String = if (nullableValue == null) "default" else nullableValue

val v: String = nullableValue ?: "default"
```

Null-Safety and Means for Null Handling

```
val city = order.customer.address.city
```

```
val city = order!!.customer!!.address!!.city
```

avoid this!

```
if (order == null || order.customer == null ||
   order.customer.address == null){
    throw IllegalArgumentException("Invalid Order")
}
val city = order.customer.address.city
```

smart-cast

```
val city = order?.customer?.address?.city
```

```
val city = order?.customer?.address?.city ?:
   throw IllegalArgumentException("Invalid Order")
```

Expressions

Flow control structures are expressions!

```
val json = """{"message": "HELLO"}"""
val message = try {
    JSONObject(json).getString("message")
} catch (ex: JSONException) {
    json
}
```

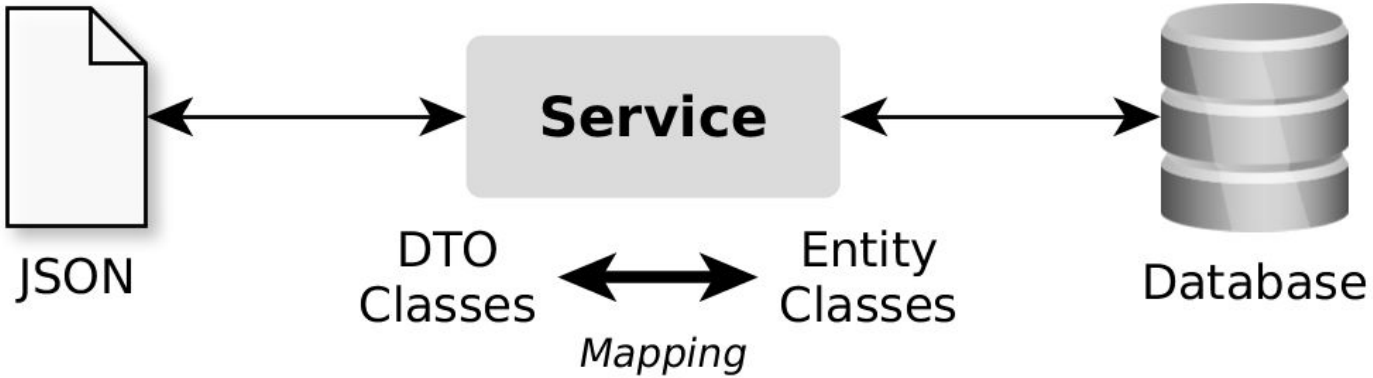
Expressions

Single Expression Functions

```
fun getMessage(json: String): String {
    val message = try {
        JSONObject(json).getString("message")
    } catch (ex: JSONException) {
        json
    }
    return message
}
```

```
fun getMessage(json: String) = try {
    JSONObject(json).getString("message")
} catch (ex: JSONException) {
    json
}
```

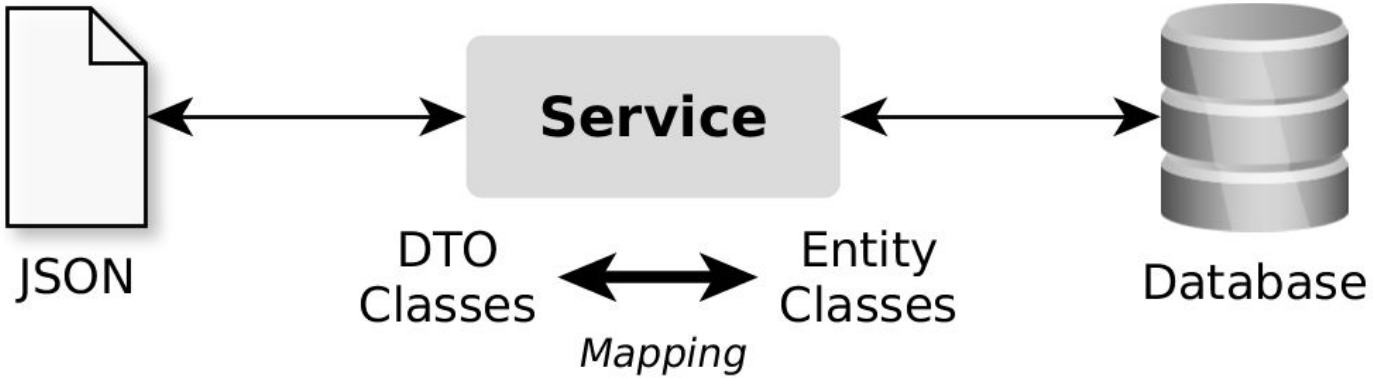
Concise Mapping between Model Classes




```
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}"
)
```

Processing an HTTP Response in Java

```
public Product parseProduct(Response response){
    if (response == null){
        throw new ClientException("Response is null");
    }
    int code = response.code();
    if (code == 200 || code == 201){
        return mapToDTO(response.body());
    }
    if (code >= 400 && code <= 499){
        throw new ClientException("Sent an invalid request");
    }
    if (code >= 500 && code <= 599){
        throw new ClientException("Server error");
    }
    throw new ClientException("Error. Code " + code);
}
```

Processing an HTTP Response in Kotlin: when

```
fun parseProduct(response: Response?) = when (response?.code()){
    null -> throw ClientException("Response is null")
    200, 201 -> mapToDTO(response.body())
    in 400..499 -> throw ClientException("Sent an invalid request")
    in 500..599 -> throw ClientException("Server error")
    else -> throw ClientException("Error. Code ${response.code()}")
}
```

Do-It-Yourself ORM

```
@Component
class UserDAO(private val template: JdbcTemplate) {

    fun findAllUsers() = template.query("SELECT * FROM users;", this::mapToUser)

    fun findUser(id: Int) = try {
        template.queryForObject("SELECT * FROM users WHERE id = $id;", this::mapToUser)
    } catch (e: EmptyResultDataAccessException) {
        null
    }

    private fun mapToUser(rs: ResultSet, rowNum: Int) = User(
            id = rs.getInt("id")
            , email = rs.getString("email")
            , name = mergeNames(rs)
            , role = if (rs.getBoolean("guest")) Role.GUEST else Role.USER
            , dateCreated = rs.getTimestamp("date_created").toInstant()
            , state = State.valueOf(rs.getString("state"))
    )
}
```

Spring: Easy Constructor Injection

```java
// Java
public class CustomerResource {

    private CustomerRepository repo;
    private CRMClient client;

    public CustomerResource(CustomerRepository repo, CRMClient client) {
        this.repo = repo;
        this.client = client;
    }
}
```

```kotlin
// Kotlin
class CustomerResource(private val repo: CustomerRepository,
                       private val client: CRMClient){
}
```

Collection API

Read-only Collections

```
val list = listOf(1,2,3,4)
list.add(1)
```

Collections API

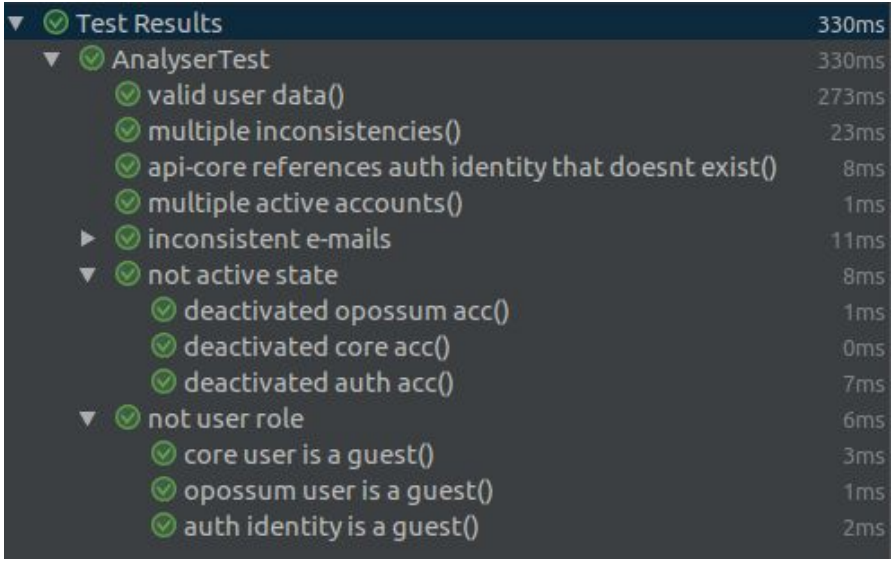
```
val evenList = list.filter { it % 2 == 0 }

val daysList = list.filter { it % 2 == 0 }
                   .map { DayOfWeek.of(it) }
println(daysList) //[TUESDAY, THURSDAY]
```

Testing: Backticks and Nested Classes

```
class AnalyserTest {
    @Test
    fun `valid user data`() {
        val inconsistencies = Analyser.find(createValidData())
        assertThat(inconsistencies).isEmpty()
    }

    @Nested
    inner class `inconsistent e-mails` {
        @Test
        fun `different auth mail`() {
        }
    }
}
```



Extension Functions

```
// definition
fun String.wrap(wrapWith: String) = wrapWith + this + wrapWith

// usage
val wrapped = "hello".wrap("*")
// as opposed to:
val wrapped = StringUtils.wrap("hello", "*")
```

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()
```

Kotlin at Spreadshirt

Ecosystem vs. Language

Evaluation of Kotlin

Pros

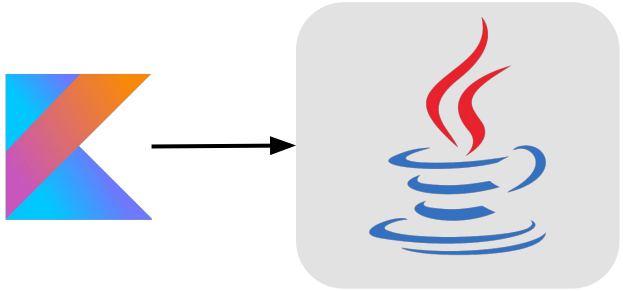
- **Reuse of the powerful and well-known Java ecosystem**
- Interoperability with Java.
- Productivity
- Less error-prone
- Easy to learn. No paradigm shift.
- Stepwise migration possible.
- Brilliant IDE support with IntelliJ IDEA.

Cons

- Training required
- **Further development depends on JetBrains.**
- Poor Support for other IDEs (like Eclipse)

⇒ Low Risks

Kotlin Usage at Spreadshirt



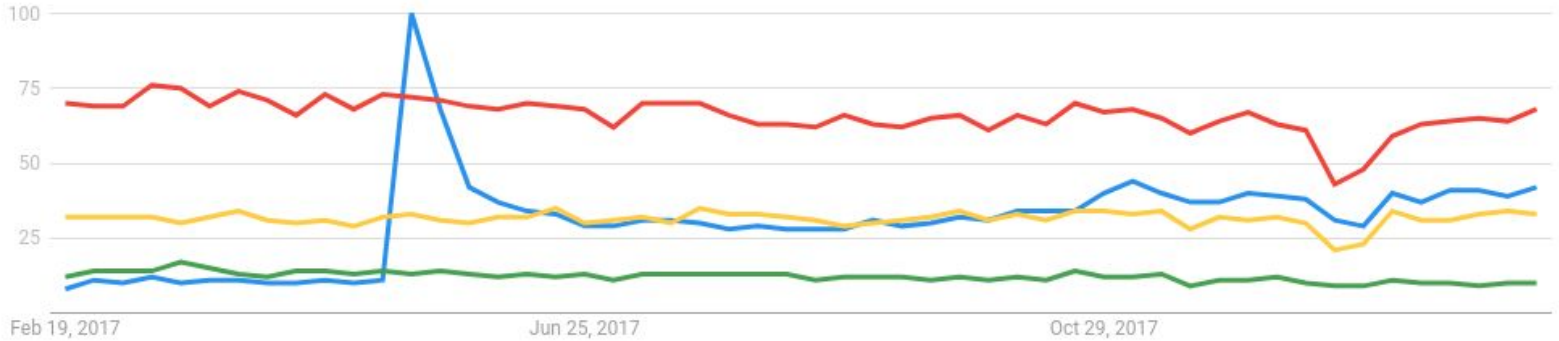
3 Test Projects

1 Java service enriched with Kotlin

13 new services and tools purely written in Kotlin

Adoption of Kotlin Today (Outside of Spreadshirt)

Google Search Trends



Scala
Kotlin
Groovy
Clojure

Peak: Google I/O '17:

"Kotlin is an official language for Android development"

Further Support and Integrations for Kotlin

- start.spring.io
- Kotlin Compiler Plugin
- Kotlin Support in Spring 5.0

- Kotlin Gradle DSL

- `@TestInstance(Lifecycle.PER_CLASS)`

- Kotlin Android Extensions

Pitfalls

Missing Default Constructor for Data Classes

```
data class Snippet(val code: String, val author: String)
val snippet = Snippet()
```

⇒ Issues with Object Mapping:

- JAXB requires default constructor ↯
- Jackson: jackson-module-kotlin allows parameterized constructors
- Hibernate: kotlin-noarg compiler plugin for JPA → Synthetic default constructor

```
apply plugin: "kotlin-jpa"
```

- Spring Data MongoDB: `@PersistenceConstructor` or kotlin-noarg plugin for `@Document`

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:
  - Interfaces
  - `Open` classes and methods explicitly
  - Spring: 'kotlin-spring' Open-all-plugin for Kotlin compiler.
  - Mockito: Use MockK instead or Mockito's Incubating Feature

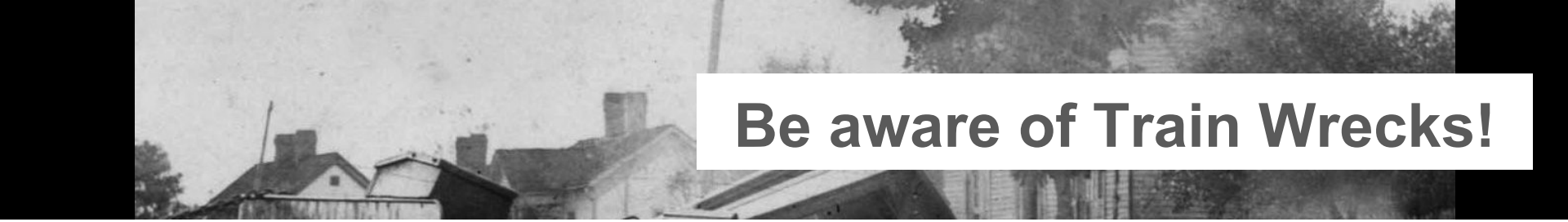
Minor Annoying Aspects of Kotlin

- Language:
  - No package-private visibility
  - No multi-catch

Annoys rarely

- Compilation Speed
  - Build is slower
  - IDE feels slower

Will become better;
Gradle: Inkr. Builds, Daemons



Be aware of Train Wrecks!

```
fun map(dto: OrderDTO, authData: RequestAuthData) = OrderEntity(
    id = dto.id,
    shopId = try {
        extractItemIds(dto.orderItems[0].element.href).shopId
    } catch (e: BatchOrderProcessingException) {
        restExc("Couldn't retrieve shop id from first order item: ${e.msg}")
    },
    batchState = BatchState.RECEIVED,
    orderData = OrderDataEntity(
        orderItems = dto.orderItems.map { dto -> mapToEntity(dto) },
        shippingType = dto.shipping.shippingType.id,
        address = mapToEntity(dto.shipping.address),
        correlationOrderId = dto.correlation?.partner?.orderId,
        externalInvoiceData = dto.externalInvoiceData?.let { ExternalInvoiceDataEntity(
                url = it.url,
                total = it.total,
                currencyId = it.currency.id
        )}
    ),
    partnerUserId = authData.sessionOwnerId ?: restExc("No sessionId supplied", 401),
    apiKey = authData.apiKey,
    dateCreated = if (dto.dateCreated != null) dto.dateCreated else Instant.now(),
)
```

Conclusion

Kotlin at Spreadshirt:
A Success Story!

Questions?