Cleaner Code with Kotlin

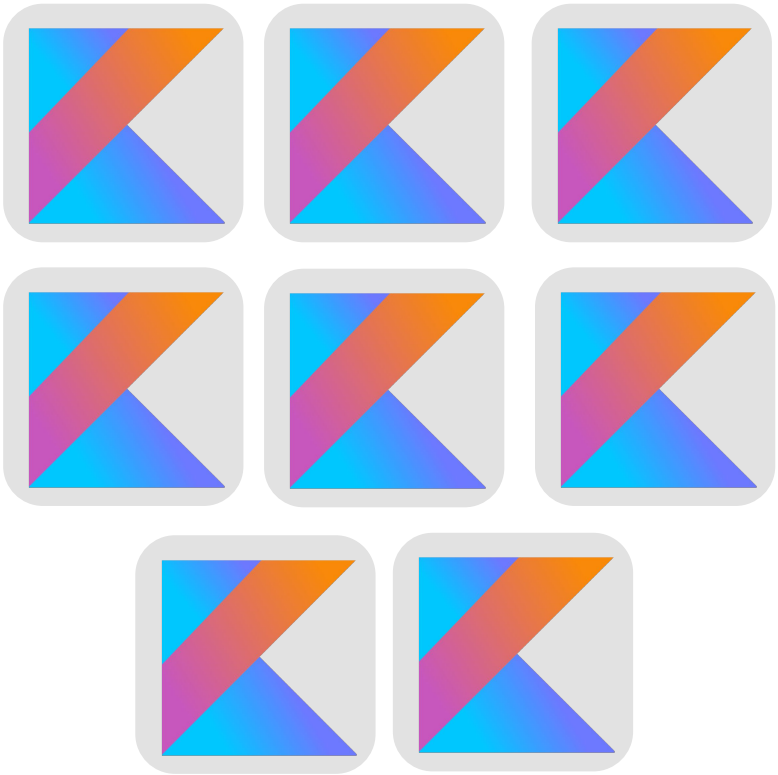
Philipp Hauer
Spreadshirt

Clean Code Days 2017

Hands Up!

Spreadshirt

Kotlin Usage at Spreadshirt



8 new services and tools purely written in Kotlin

1 Java service enriched with Kotlin

Recap:
What is Clean Code?

Recap: What is Clean Code?

readable

intuitive

concise

short

easy to understand

simple

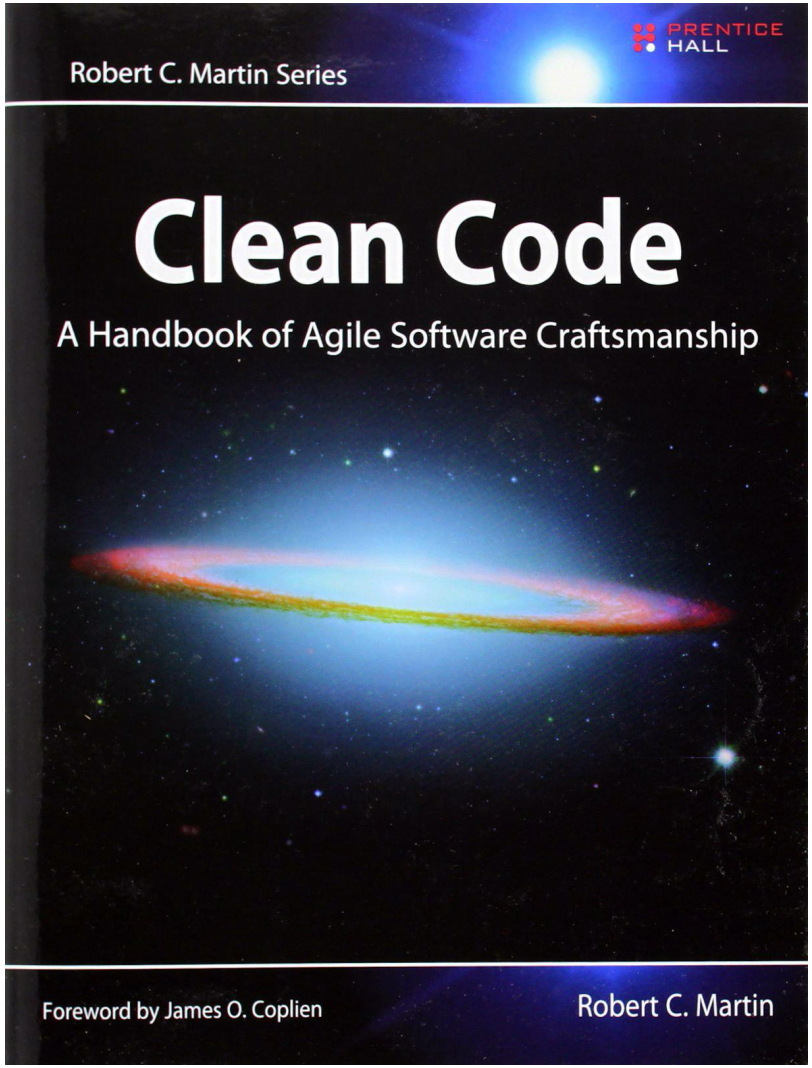
expressive

minimal ceremony

minimal syntactic noise

Clean Code and Kotlin

Clean Code and Kotlin



Functions: “Small” Functions with Java

"Rule 1: Functions should be small!
Rule 2: Functions should be smaller than that!"

```
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);
}
```

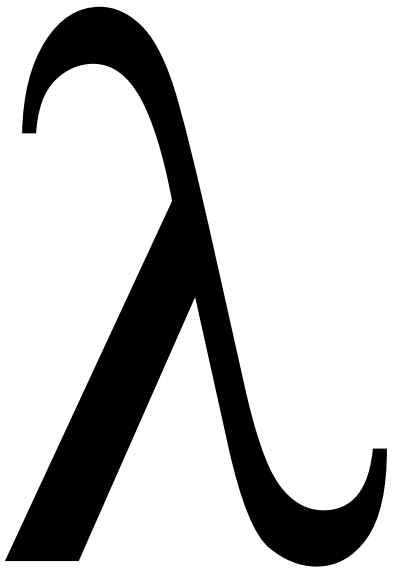
Functions: Small Functions with Kotlin

```
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()}")
}
```

Functions: Side-Effects

“Reduce side-effects!”

“No unexpected and hidden changes!”



Better Support for Functional Programming in Kotlin

- **Expressions**
- **Immutability**
- Function Types
- Concise Lambda Expressions
- Kotlin’s Collection API

Expressions in Kotlin

Flow control structures are expressions!

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

Expressions in Kotlin

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
}
```

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(),
)
```

Immutability: Feels Natural and Easy

Immutable References

```
val id = 1
id = 2
var id2 = 1
id2 = 2
```

Read-only Collections

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

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

Immutability: Feels Natural and Easy

Immutable Data Classes

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

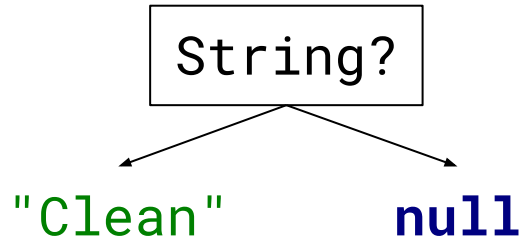
- Constructor (assign args to props)
- Getter
- toString()
- hashCode(), equals()
- copy()
- final
- Default Arguments (no chaining)

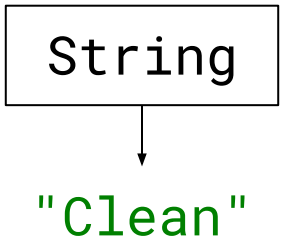
```
val design = DesignData(id = 1, fileName = "cat.jpg",
   uploaderId = 2)

val id = design.id
design.id = 2

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

Error Handling

Clean Code Recommendation	Kotlin Support?
Separate error handling from logic	/
Use exceptions instead of returning null	/
Don’t use checked exceptions	Checked exceptions don’t exist.
Strategies to avoid null	/
Don’t return null, because:	/
a) Scattered code with null-checks	Concise syntax for dealing with null.
b) Easy to forget null-check. NPE.	Nullable types. Compiler enforces handling.

Nullability in Kotlin






```
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"
```

Nullability in Kotlin

```
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

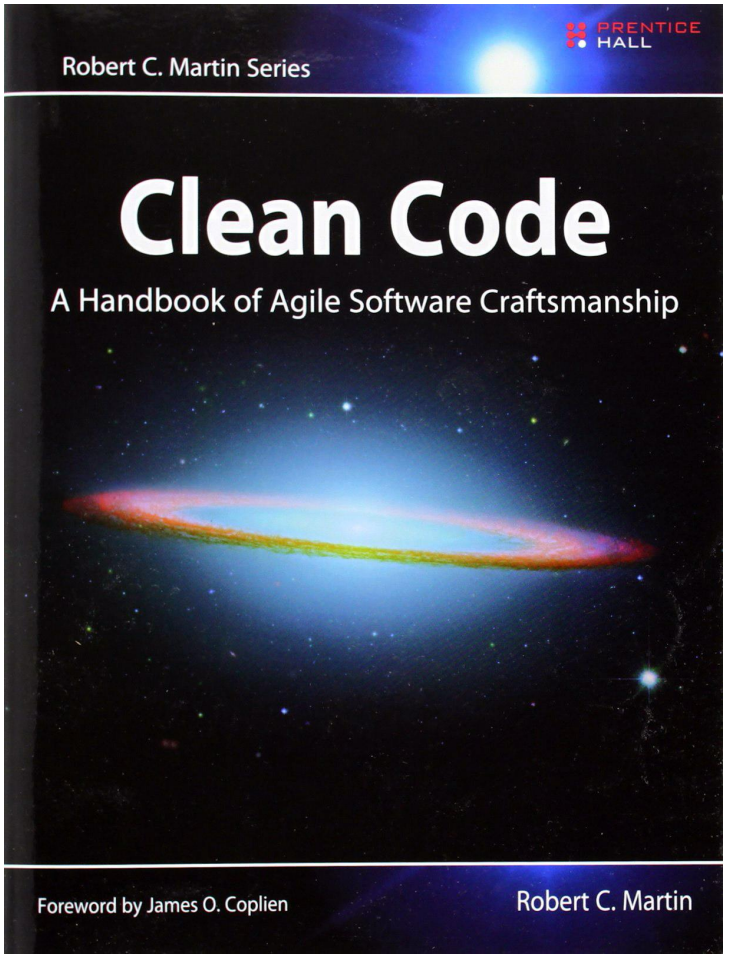
```
val city = order?.customer?.address?.city
```

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

Restrictions

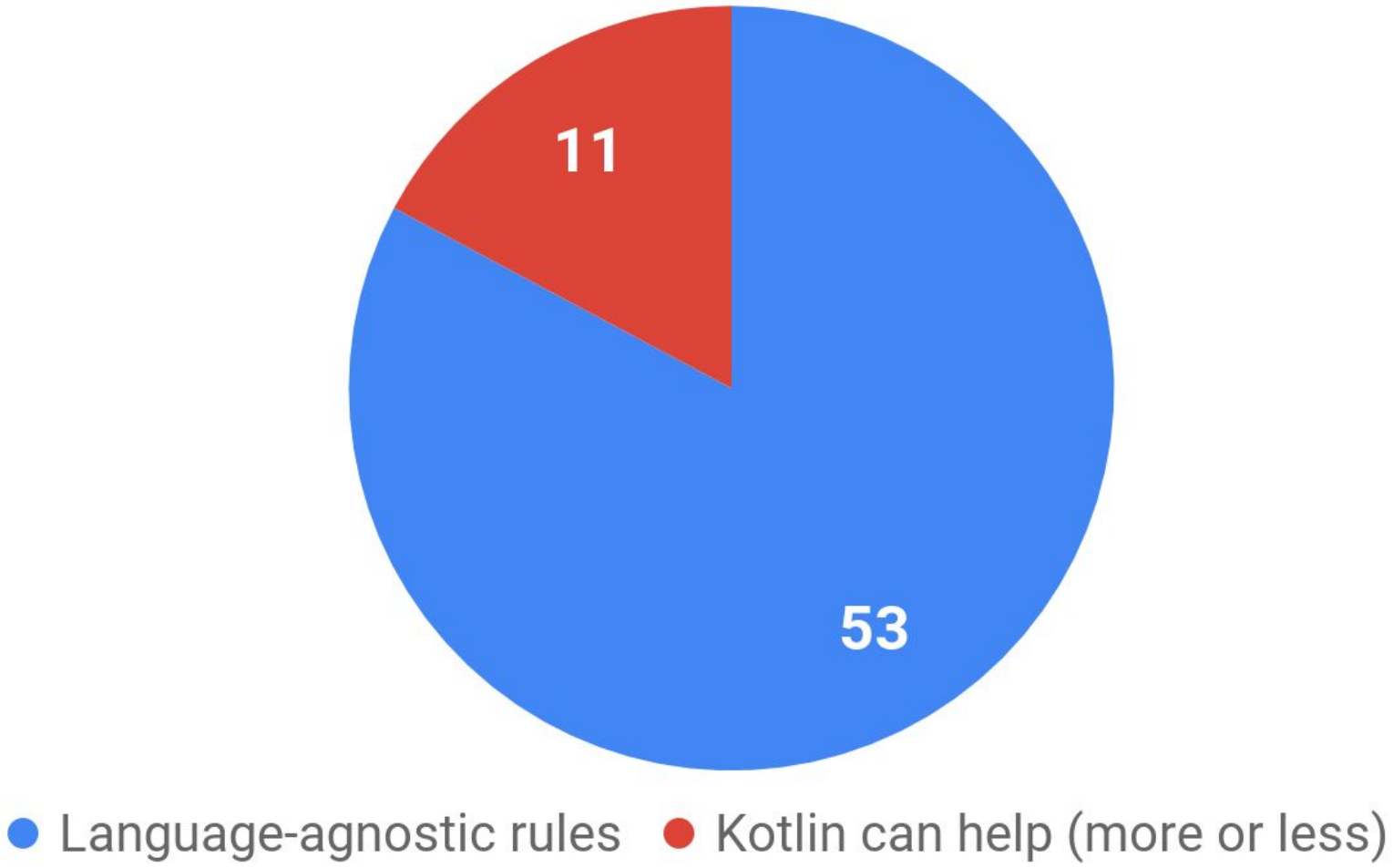
Clean Code Chapters

- Meaningful Names
- **Functions**
- Comments
- Formatting
- **Object and Data Structures**
- **Error Handling**
- Boundaries
- Classes
- Systems
- Emergence
- **Concurrency**



⇒ **Kotlin can help for 4 of 11 items**

Clean Code: Smells and Heuristics



Hammer and Nails

Be carefull with:

- Unreadable monster expressions
- Complicated null-safe-calls and elvis structures

```
//Don't
value?.emptyToNull()?.let { map.put("bla", it) }
```

```
fun  String.emptyToNull() =
    if (this.isEmpty()) null
    else this
```

```
//Don't
if (value?.isNotEmpty() ?: false){
    map.put("key", value!!)
}
```

```
// KISS!
if (!value.isNullOrEmpty()){
    map.put("key", value!!)
}
```

Readability and Simplicity is (still) King!

Conclusion

Cleaner Code with Kotlin?

Yes!

- Less boilerplate and syntactic noise → readability
- Safer
- Kotlin encourages good design

But:

- Clean code and good design is no automatism with Kotlin! Developer's discipline is still important!
- Use some features with sound judgement. *"Clarity is King"*

Thanks!
Let’s get in touch!

Twitter: @philipp_hauer

Blog: blog.philipphauer.de