Extension Methods in Scala and Kotlin

By Brandon Raboy

Structure and Thesis

1) History of Programming Languages
2) Scala Features
3) Kotlin Features
4) Extension Methods in Scala and Kotlin

ALGOL 60 Report

- Nested block structures
- Lexical scoping
- Backus-Naur form (BNF)

ALGOL 68

- Anonymous routines and recursive typing system with high-order functions
- Van Wijngaarden grammar

Scala's key features

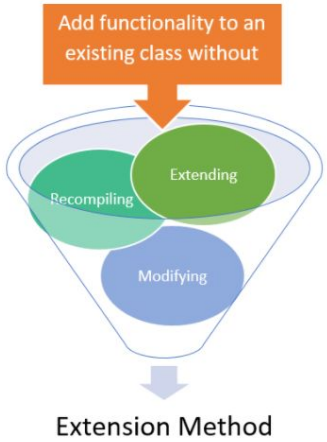
- syntactic flexibility
- unified type system
- for-expressions
- functional tendencies
- object-oriented extensions
- an expressive type system
- type enrichment

Kotlin’s key features

- Semicolons optional
- Data type requirements
- val and var keywords
- Classes are final
- Class members are public

What are Extension Methods?

- methods added to objects after the original objects are compiled



Defining an Extension Method in Scala

```
object StringExtensions {
  extension (str: String) {
    def toSnakeCase = {
      str.replaceAll("([A-Z])", "_" + "$1").toLowerCase
    }
  }
}
```

Multiple extension methods in Scala

```
object StringExtensions {
            extension (str: String) {
                    def toSnakeCase = {
            str.replaceAll("([A-Z])", "_" + "$1").toLowerCase
                    }
                    def isNumber = {
        str.matches("[0-9]+")
                    }
            }
}
```

Generic Extensions in Scala

```
object GenericExtensions {
	extension [T](list: List[T]) {
		def getSecond = if(list.isEmpty) None
else list.tail.headOption
	}
}
```

Type constraints in Scala

```
extension [T: Numeric](a: T) {
      def add(b:T): T = {
            val numeric = summon[Numeric[T]]
            numeric.plus(a, b)
      }
}
```

Defining an extension method in Kotlin

```
fun String.escapeForXml() : String {
  return this
    .replace("&", "&amp;")
    .replace("<", "&lt;")
    .replace(">", "&gt;")
}
```

Generic Extensions in Kotlin

```
fun <T> T.concatAsString(b: T) : String {
    return this.toString() + b.toString()
}
```

Infix extension methods in Kotlin

```
infix fun Number.toPowerOf(exponent: Number): Double {
    return Math.pow(this.toDouble(), exponent.toDouble())
}
```

To call this method

```
3 toPowerOf 2 // 9
9 toPowerOf 0.5 // 3
```

Shorthand operator method in Kotlin

```
operator fun List<Int>.times(by: Int): List<Int> {
    return this.map { it * by }
}
```

To call this method:

```
listOf(1, 2, 3) * 4 // [4, 8, 12]
```