Extension methods

An Example

Introduce this declaration at **file-level scope** (not in any class)

```
fun String.count(c: Char): Int = this.count { it == c }
```

Indicates that the method is being added to String – the receiving object of a call to count will be a String

`this` refers to the string that receives the call

When there is no ambiguity, this can be omitted :

```
fun String.count(c: Char): Int = count { it == c }
```

Another Example

```
operator fun String.times(count: Int): String = repeat(count)
operator fun Int.times(toBeRepeated: String): String =
toBeRepeated.repeat(this)
```

Alternative:

```
operator fun Int.times(toBeRepeated: String): String = toBeRepeated
* this
```

An extra method call, but easier to read, with less code duplication.

Extension methods on generic classes

```
fun <A, B> Pair<A, B>.equalComponents() = first == second
```

This can be done between two unknown, possibly different types, since `.equals()` is defined in the `Any` class. **Equality is always available.**

Extending a generic class for specific types

```
operator fun Pair<Double, Double>.plus(
    other: Pair<Double, Double>,
): Pair<Double, Double> =
    Pair(first + other.first, second + other.second)
```

An extension method inside a class

Let’s make a redacting string builder – checks each string that is passed to append against a list of bad words.

Example Usage

```
val builder = RedactingStringBuilder(setOf("Haskell", "monad",
"category", "functor"))
builder.append("My")
builder.append(" ")
builder.append("favourite")
builder.append(" ")
builder.append("programming")
builder.append(" ")
builder.append("language")
builder.append(" ")
builder.append("is")
builder.append(" ")
builder.append("Haskell")
builder.append(" ")
println(builder.toString())
```

Output:

```
My favourite programming language is _______
```

```
class RedactingStringBuilder(private val badWords: Set<String>) {
    private val stringBuilder = StringBuilder()
    val length: Int = stringBuilder.length
    fun append(text: String) = stringBuilder.append(text.redact())
```

```
    override fun toString(): String {
        return stringBuilder.toString()
    }
    fun String.redact(): String =
        if (this in badWords) {
            "_".repeat(length)
            // This is the length of the string, because the
receiver                of this method is a String
            // The length property of RedactingStringBuilder is
shadowed
        } else {
            this
        }
}
```

This extension can only be called by a function that has `RedactingStringBuilder` as its receiver.

To refer to enclosing class's version of a shadowed property, use `this@EnclosingClassName`.

```
class RedactingStringBuilder(private val badWords: Set<String>) {
    private val stringBuilder = StringBuilder()
    val length: Int = stringBuilder.length
    ...
    fun String.redact(): String {
        // Provides access to length of the enclosing
RedactingStringBuilder
        println(this@RedactingStringBuilder.length)
        if (this in badWords) {
            return "_".repeat(length)
            // Provides access to length of the receiving String -
equivalent to this.length
        }
        return this
    }
}
```

Can we invoke such an extension method from outside the class?

```
fun main() {
    val builder = RedactingStringBuilder(setOf(
        "Haskell",
        "monad",
        "category",
        "functor"),
    )
    with (builder) {
        println("Haskell".redact())
    }
}
```

The `with` statement makes the target object `this` for the scope that follows.

All methods of `RedactingStringBuilder` are in scope here, including its `redact` extension to `String`.

Extension methods can help with look and feel, but are really just syntactic sugar. Extension method do not get access to private properties and methods.

An extension method only has access to the service provided by the class it extends – the public properties and methods. An extension method is a client of the class it extends. An extension method adds convenience services to the class for other clients to use.