Methodology for Writing Code in the Kotlin Language

This methodology contains current coding style in the Kotlin language used by the Kill Bills team.

Naming Style

If in doubt, default to the Java Coding Conventions such as:

- use of camelCase for names (and avoid underscore in names)
- types start with upper case
- methods and properties start with lower case
- use 4 space indentation
- public functions should have documentation such that it appears in Kotlin Doc

Colon

There is a space before colon where colon separates type and supertype and there's no space where colon separaters instance and type:

```
interface Foo<out T : Any> : Bar {
    fun foo(a: Int): T
}
```

Lambdas

In lambda expressions, spaces should be used around the curly braces, as well as around the arrow which separates the parameters from the body. Whenever possible, a lambda should be passed outside of parentheses.

```
list.filter { it > 10 }.map { element -> element * 2
}
```

In lambdas which are short and not nested, it's recommended to use the it convention instead of declaring the parameter explicitly. In nested lambdas with parameters, parameters should be always declared explicitly.

Class header formatting

Classes with a few arguments can be written in a single line:

```
class Person(id: Int, name: String)
```

Classes with longer headers should be formatted so that each primary constructor argument is in a separate line with indentation. Also, the closing parenthesis should be on a new line. If we use inheritance, then the superclass constructor call or list of implemented interfaces should be located on the same line as the parenthesis:

```
class Person(
    id: Int,
    name: String,
    surname: String
) : Human(id, name) {
    // ...
}
```

For multiple interfaces, the superclass constructor call should be located first and then each interface should be located in a different line:

```
class Person(
    id: Int,
    name: String,
    surname: String
) : Human(id, name),
    KotlinMaker {
    // ...
}
```

Constructor parameters can use either the regular indent or the continuation indent (double the regular indent).

Unit

If a function returns Unit, the return type should be omitted:

```
fun foo() { // ": Unit" is omitted here

}
```

Functions vs Properties

In some cases functions with no arguments might be interchangeable with read-only properties. Although the semantics are similar, there are some stylistic conventions on when to prefer one to another.

Prefer a property over a function when the underlying algorithm:

- does not throw any Exceptions
- has a O(1) complexity
- is cheap to calculate (or cached on the first run)
- return the same result over invocations