**Определение**

Hard coding -- это практика разработки программного обеспечения, заключающаяся во встраивании данных непосредственно в исходный код программы или другого исполняемого объекта

**Когда появляется:**

Этот анти-паттерн тесно связан с магическими числами и магическими строчками

**Пример:**

Плохо:

**class** **User** {

**fun** **checkName**(name: **String**) {

**if** (name.length > 42) {

**throw** **IllegalArgumentException**("username is too long")

}

*// ...*

}

}

Хорошо:

**class** **User** {

**fun** **checkName**(name: **String**) {

**if** (name.length > **MAX\_USERNAME\_SIZE**) {

**throw** **IllegalArgumentException**("username is too long")

}

*// ...*

}

**companion** **object** {

**private** **const** **val** MAX\_USERNAME\_SIZE = 42

}

}

Проблема с кроссплатформенность:

**const** size\_t NUMBER\_OF\_ELEMENTS = 10;

long a[NUMBER\_OF\_ELEMENTS];

memset(a, 0, 10 \* 4); *// неправильно — подразумевается, что long равен 4 байтам, используется магическое число элементов*

memset(a, 0, NUMBER\_OF\_ELEMENTS \* 4); *// неправильно — подразумевается, что long равен 4 байтам*

memset(a, 0, NUMBER\_OF\_ELEMENTS \* **sizeof**(long)); *// не совсем правильно — дублирование имени типа (если изменится тип, то придется менять и здесь)*

memset(a, 0, NUMBER\_OF\_ELEMENTS \* **sizeof**(a[0])); *// правильно, оптимально для динамических массивов ненулевого размера*

memset(a, 0, **sizeof**(a)); *// правильно, оптимально для статических массивов*

Иногда использование магических чисел оправдано:

1. 0 и 1 для инициализации и инкрементирования значений в циклах

for (x in 0..max) *println*(x) *// Prints 0 through max (inclusive)*

1. 2 для проверки четности

val isEven: (Int) -> Boolean = **{ it** % 2 == 0 **}**

1. Простые арифметические константы

val circumference = 2 \* Math.*PI* \* radius

val d = b \* b - 4 \* a \* c

Плохо:

fun buildConnection(): Connection? {

return DriverManager.getConnection(

"jdbc:mysql://localhost:3306/someDb?characterEncoding=UTF-8&characterSetResults=UTF-8&serverTimezone=UTC",

"user01",

"12345qwert")

}

Хорошо:

Выносим в конфиг

spring:

datasource: jdbc-url:jdbc:mysql://localhost:3306/someDb?characterEncoding=UTF-8

username: user01

password: 12345qwert

Плохо:

layout/MainActivity.xml

...

<**TextView**

**android:id="@+id/textView"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="Все любят котиков"**

...

/>

...

Хорошо:

layout/MainActivity.xml

...

<**TextView**

**android:id="@+id/textView"**

**android:layout\_width="wrap\_content"**

**android:layout\_height="wrap\_content"**

**android:text="@string/important\_text"**

...

/>

...

values/strings.xml

<**resources**>

...

<**string name="important\_text"**>Все любят котиков</**string**>

...

</**resources**>

**Какие проблемы несет:**

1. Трудно читать и понимать
2. Код будет исправно работать только в окружении, в котором ведется разработка.
3. Может проявляется непредсказуемые дефекты во время перемещения, переименования файлов, и их поведение может меняться при изменения конфигурации устройств.

**Как избежать:**

1. Выносить магические числа и строки в константы или конфиги
2. Стараться писать максимально обобщенный код (но не увлекаться, иначе получим soft coding)