Лекция #7. Тип null в Kotlin

У всех переменных и констант, с которыми мы работали до сих пор, были конкретные значения. У переменной типа **string**, вроде **var name**, есть строковое значение, которое с ней ассоциируется. К примеру, "Joe Howard". Это может быть и пустая строка вроде "", однако и в ней все же есть значение, к которому можно отсылаться.

Это одна из встроенных особенностей для безопасности в **Kotlin**. Если используется тип **Int** или **String**, тогда целое число или строка действительно существует — гарантированно.

Основы использования Null в Kotlin

Иногда полезно иметь возможность показать отсутствие значения. Представьте случай, когда вам нужно сделать ссылку на персональные данные человека: требуется сохранить его имя, возраст и род занятий. У всех людей есть имя и возраст, так что эти значения определенно будут присутствовать. Однако не у всех есть работа, поэтому важно, чтобы можно было не указывать род занятий.

Допустим, мы не знаем про **null**, тогда имя, возраст и род занятий человека можно было бы указать следующим образом:

```
1  var name = "Джо Говард"
2  var age = 24
3  var occupation = "Программист и Автор"
```

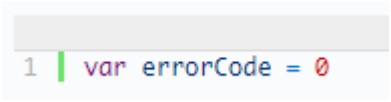
Но что, если человек стал безработным? Может, он достиг просвещения и решил жить на вершине горы. Здесь пригодилась бы возможность указать на отсутствие значения.

Почему бы просто не использовать пустую строку? Это можно сделать, однако **nullable** типы будут наиболее оптимальным решением. Далее подробнее о том, почему именно.

Сторожевое значение (Sentinel) в Kotlin

Действительное значение, которое представляет специальное условие вроде отсутствия значения, называется **сторожевое значением**. В предыдущем примере это была бы пустая строка.

Давайте рассмотрим другой пример. Допустим, код запрашивает что-то с сервера, и вы используете переменную для хранения любого возвращаемого кода ошибки:



В случае успеха, отсутствие ошибки обозначается нулем. Это означает, что 0 является сторожевым (**sentinel**) значением. Как и пустая строка для названия рода деятельности, это работает, но может сбить с толку программиста. 0 на самом деле может быть допустимым кодом ошибки — или может быть в будущем, если сервер изменит свой ответ. В любом случае нельзя быть полностью уверенным в том, что сервер не вернул ошибку, не ознакомившись с документацией. В этих двух примерах было бы лучше, если бы существовал специальный тип, который мог бы представлять отсутствие значения. Тогда было бы понятно, когда значение существует, а когда нет.

Null является названием, которое дается отсутствию значения. Мы рассмотрим, как **Kotlin** напрямую внедряет данный концепт в язык довольно элегантным способом. Некоторые другие языки программирования просто используют сторожевые значения. В других есть концепт **null** значения, но это только синоним для нуля.

Kotlin вводит целый новый набор типов — **nullable** типы. Они позволяют значению быть равным **null**. Если вы обрабатываете не-**null** тип, у вас точно будут какие-то значения, поэтому не нужно беспокоиться о существовании действительного значения. Аналогично, при использовании **nullable** типа вы должны обрабатывать случаи использования типа **null**. Так убирается двусмысленность, присущая использованию **sentinel** значениям.

Основы использования nullable типов

Nullable типы в **Kotlin** являются решением проблемы представления значения или его отсутствия. **Nullable** типы могут содержать какое-то значение или содержать **null**.

Концепт **nullable** можно сравнить с ящиком: в нем или есть значение, или его нет. Если значения нет, он содержит **null**. Ящик сам по себе существует всегда. Вы всегда можете его открыть и посмотреть, что внутри.

С другой стороны, у типов **String** или **Int** нет такого ящика. Вместо этого у них в любом случае будет какое-то значение, к примеру, "hello" или 42. Помните, что у не-**null** типов гарантированно есть действительное значение.

Переменная **nullable** типа объявляется через использование следующего синтаксиса:

```
1 var errorCode: Int?
```

Единственная разница между таким и стандартным способом объявлением переменной, это наличие вопросительного знака после типа. В данном случае **errorCode** является «**nullable Int**». Это значит, что сама переменная похожа на ящик, содержащий либо **Int**, либо **null**.

Также обратите внимание, что **nullable** тип должен создаваться явным способом с помощью объявления типа (например: **Int?**). **Nullable** типы **никогда не могут быть выведены из значений инициализации**, поскольку у этих значений обычный не-**null** тип.

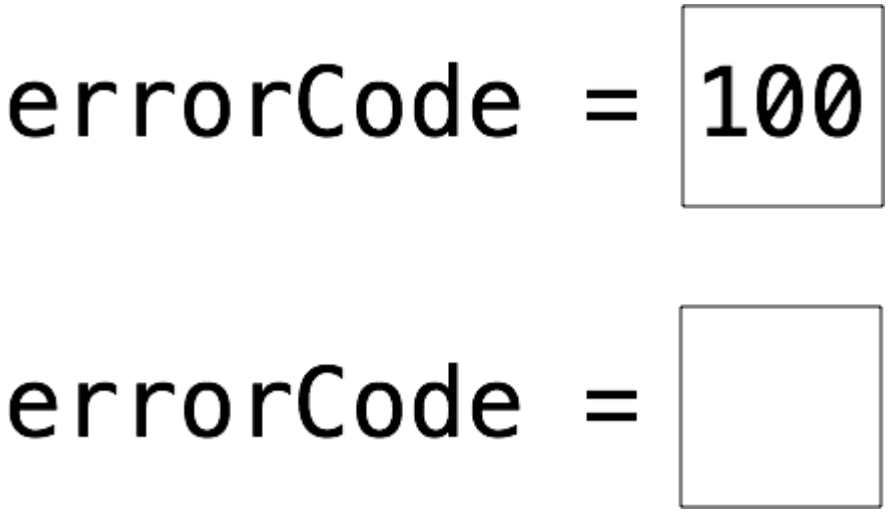
Указать значение для такой переменной очень просто. Значение **Int** можно назначить следующим образом:

```
1 errorCode = 100
```

Или вы можете указать значение **null**:

```
1 errorCode = null
```

Следующая диаграмма может помочь визуализировать, что происходит в обоих вариантах:



Так называемый **nullable** ящик существует всегда. Во время присваивания переменной значения 100 вы заполняете ящик значением. Когда вы присваиваете переменной значение **null**, данный ящик опустошается.