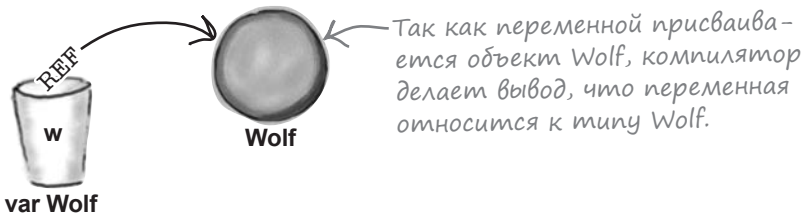
Как удалить ссылку на объект из переменной?

Вы уже знаете, что, если требуется определить новую переменную `Wolf` и присвоить ей ссылку на объект `Wolf`, это можно сделать следующей командой:

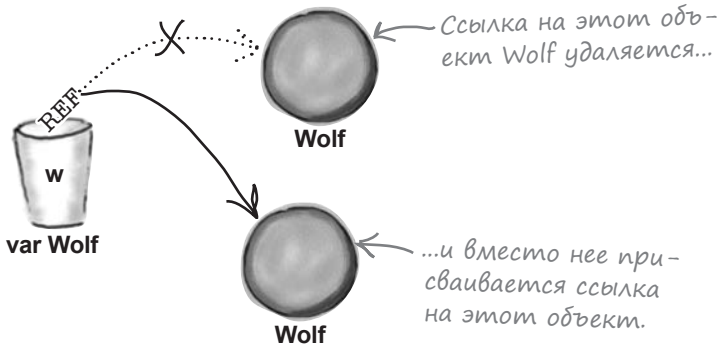
```
var w = Wolf()
```

Компилятор видит, что вы присваиваете объект `Wolf` переменной `w`, и поэтому делает вывод, что переменная должна иметь тип `Wolf`:



После того как компилятор определит тип переменной, он следит за тем, чтобы в ней хранились *только* ссылки на объекты `Wolf` и любые подклассы `Wolf`. Таким образом, если переменная определена с ключевым словом `var`, ее значение можно обновить так, чтобы в ней хранилась ссылка на другой объект `Wolf`, например:

```
w = Wolf()
```



А если вы хотите обновить переменную так, чтобы в ней не хранилась ссылка *ни на какой объект*? **Как удалить ссылку на объект из переменной после того, как вы выполнили присваивание?**

Удаление ссылки на объект с использованием null

Чтобы удалить ссылку на объект из переменной, присвойте переменной значение **`null`**:

```
w = null
```

Значение null означает, что переменная не содержит ссылку на какой-либо объект: переменная существует, но ни на что не указывает.

Но тут снова возникает Коварная Ловушка. По умолчанию *типы Kotlin не поддерживают значения null.* **Если вам нужна переменная, способная хранить null, вы должны явно указать, что ее тип является null-совместимым.**

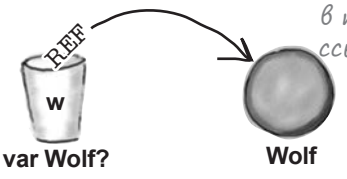
Для чего нужны null-совместимые типы?

null-совместимый тип способен хранить значения null. В отличие от других языков, Kotlin следит за значениями, которые могут быть равны null, чтобы вы не пытались выполнять с ними недопустимые операции. Выполнение недопустимых операций со значениями null — это самая распространенная причина ошибок времени выполнения в таких языках, как Java. Они могут вызвать сбой в вашем приложении. Однако в Kotlin такие проблемы встречаются редко благодаря умному использованию null-совместимых типов.

Чтобы объявить тип null-совместимым, поставьте после него вопросительный знак (`?`). Например, для создания null-совместимой переменной `Wolf` и присваивания ей нового объекта `Wolf` используется следующий код:

```
var w: Wolf? = Wolf()
```

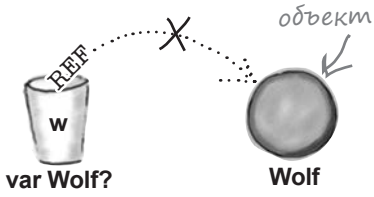
Запись Wolf? означает, что в переменной могут храниться ссылки на объекты Wolf или null.



А если вы хотите удалить ссылку на `Wolf` из переменной, используйте следующую команду:

```
w = null
```

Когда вы присваиваете w значение null, ссылка на объект Wolf удаляется.



Где же используются null-совместимые типы?

Мысл null

Когда вы присваиваете переменной `null`, происходит примерно то же, что при стирании программы на пульте дистанционного управления. У вас есть пульт (переменная), но он не связан с телевизором (объект).

Ссылка `null` содержит набор битов, представляющих «неопределенное значение», но мы не знаем и не хотим знать, что это за биты. Система автоматически делает это за нас.

Если вы попытаетесь выполнить недействительную операцию с null в Java, произойдет печально известное исключение NullPointerException. Исключение — предупреждение о том, что в программе произошло что-то особенно неприятное. Исключения будут более подробно рассмотрены позднее в этой главе.

«null-совместимым» называется тип, способный хранить значения null наряду со своим базовым типом. Например, переменная Duck? может хранить объекты Duck и null.

null-совместимые типы могут использоваться везде, где могут использоваться не-null-совместимые

Любой тип, который вы определяете, можно преобразовать в null-совместимую версию этого типа, просто добавив ? после имени. null-совместимые типы могут использоваться везде, где могут использоваться обычные (не-null-совместимые) типы:

- **При определении переменных и свойств.**
  Любая переменная или свойство могут быть null-совместимыми, но вы должны явно определить их таковыми простым объявлением их типа с ?. Компилятор не может сам определить, какой тип является null-совместимым, и по умолчанию всегда создает не-null-совместимый тип. Таким образом, если вы хотите создать null-совместимую переменную с именем `str` и присвоить ей значение «Pizza», вы должны объявить ее с типом `String?`:

```
var str: String? = "Pizza"
```

  Обратите внимание: переменные и свойства могут инициализироваться значением `null`. Например, следующий код компилирует и выводит текст "null":

```
var str: String? = null
println(str)
```

Не путайте с командой var str: String? = "". "" — объект String, не содержащий ни одного символа, тогда как null не является объектом String.

- **При определении параметров.**
  Любую функцию или параметр конструктора можно объявить с null-совместимым типом. Например, следующий код определяет функцию с именем `printInt`, которая получает параметр типа `Int?` (null-совместимый `Int`):

```
fun printInt(x: Int?) {
    println(x)
}
```

  При определении функции (или конструктора) с null-совместимым параметром при вызове функции вы все равно должны предоставить значение этого параметра, даже если это `null`. Как и в случае с не-null-совместимыми типами параметров, этот параметр нельзя пропустить при вызове, если только для него не определено значение по умолчанию.

- **При определении возвращаемых типов функций.**
  Функция может иметь null-совместимый возвращаемый тип. Например, следующая функция имеет возвращаемый тип `Long?`:

```
fun result() : Long? {
    //Код вычисляет и возвращает Long?
}
```

Функция должна возвращать значение типа Long или null.

Также можно создавать массивы null-совместимых типов. Посмотрим, как это делается.

Как создать массив null-совместимых типов

У массива null-совместимых типов элементы могут принимать значение null. Например, следующий код создает массив с именем `myArray`, в котором хранятся элементы `String?` (null-совместимые `String`):

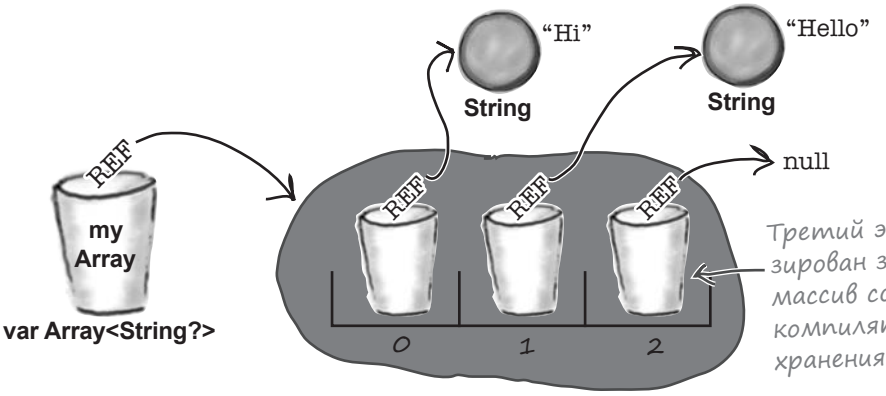
```
var myArray: Array<String?> = arrayOf("Hi", "Hello")
```

Array<String?> может хранить String и null.

Однако компилятор может прийти к выводу, что массив должен содержать null-совместимые типы, если инициализируется одним или несколькими значениями `null`. Таким образом, когда компилятор обрабатывает следующий код:

```
var myArray = arrayOf("Hi", "Hello", null)
```

он понимает, что массив может содержать сочетание `String` и `null`, и заключает, что массив должен иметь тип `Array<String?>`:



Третий элемент был инициализирован значением null. Так как массив содержит и String, и null, компилятор создает массив для хранения String?.

Вы научились определять null-совместимые типы. Теперь посмотрим, как обращаться к функциям и свойствам этих объектов.

Часто Задаваемые Вопросы

В: Что произойдет, если я инициализирую переменную значением null и предложу компилятору определить ее тип самостоятельно? Например:

```
var x = null
```

В: Компилятор видит, что переменная должна хранить значения null, но так как у него нет информации о других видах объектов, которые могут храниться в переменной, он создает переменную, способную хранить только `null`. Скорее всего, это не то, на что вы рассчитывали, поэтому если вы собираетесь инициализировать переменную значением null, обязательно укажите ее тип.

В: В предыдущей главе вы сказали, что любой объект является подклассом `Any`. Может ли переменная с типом `Any` хранить значения null?

О: Нет. Если вам нужна переменная, в которой могут храниться объекты любых типов и null, она должна иметь тип `Any?`. Пример:

```
var z: Any?
```