**Immutable-типы**

За время работы на Java вы уже, наверное, привыкли к тому, что программист практически полностью управляет состоянием всех объектов. Захотел — создал объект Cat. Захотел — переименовал его. Захотел — поменял возраст, или еще что-нибудь. Но в Java есть несколько типов данных, которые отличаются особым состоянием. Они являются **неизменяемыми**, или **Immutable**. Это значит, что если класс неизменяемый, состояние его объектов изменить невозможно. Примеры? Возможно вы удивитесь, но самый известный пример *Immutable* - класса — **String**! Казалось бы, разве мы не можем изменить значение строки? Ну, давате попробуем:

**public** **static** **void** main(String[] args) {

String str1 = "I love Java";

String str2 = str1;//обе переменные-ссылки указывают на одну строку.

System.out.println(str2);

str1 = "I love Python";//но поведение str1 никак не влияет на str2

System.out.println(str2);//str2 продолжает указывать на строку "I love Java", хотя str1 уже указывает на другой объект

}

**Вывод: *I love Java I love Java***

После того, как мы написали:

str1 = "I love Python";

объект со строкой "I love Java" не изменился и никуда не делся. Он благополучно существует и имеет внутри себя ровно тот же текст, что и раньше. Код:

str1 = "I love Python";

просто создан еще один объект, и теперь переменная str1 указывает на него. Но на объект "I love Java" мы никак не можем повлиять. Так, ладно, давайте попробуем по-другому! В классе String полно методов, и некоторые из них, похоже с виду меняют состояние строки! Вот, например, есть метод replace(). Давате поменяем слово “Java” на слово “Python” в нашей строке!

**public** **static** **void** main(String[] args) {

String str1 = "I love Java";

String str2 = str1;//обе переменные-ссылки указывают на одну строку.

System.out.println(str2);

str1.replace("Java", "Python");//попробуем изменить состояние str1, заменив слово "Java" на “Python”

System.out.println(str2);

}

**Вывод: *I love Java I love Java***

Снова не получилось! Может, метод кривой, не работает? Попробуем другой. Вот, например, substring(). Обрезает строку по номерам переданных символов. Давате обрежем нашу до первых 10 символов:

**public** **static** **void** main(String[] args) {

String str1 = "I love Java";

String str2 = str1;//обе переменные-ссылки указывают на одну строку.

System.out.println(str2);

str1.substring(10);//обрезаем исходную строку

System.out.println(str2);

}

**Вывод: *I love Java I love Java***

Ничего не поменялось. И не должно было. Как мы и сказали — объекты String неизменяемые. А что же тогда все эти методы класса String? Они же могут обрезать строку, изменить в ней символы и прочее. Зачем они тогда нужны, если ничего не происходит? Могут! Но они при этом каждый раз возвращают новый объект строки. Бесполезно писать:

str1.replace("Java", "Python");

— вы не измените исходный объект.

Но если записать результат работы метода в новую переменную-ссылку, сразу увидим разницу!

**public** **static** **void** main(String[] args) {

String str1 = "I love Java";

String str2 = str1;//обе переменные-ссылки указывают на одну строку.

System.out.println(str2);

String str1AfterReplacement = str1.replace("Java", "Python");

System.out.println(str2);

System.out.println(str1AfterReplacement);

}

Только так все эти методы String и работают. С объектом "I love Java" ничего сделать нельзя. Только создать новый объект, и написать: “Новый объект = результат каких-то манипуляций с объектом "I love Java"”.

**Какие типы еще относятся к Immutable?** Из того, что вам железобетонно нужно запомнить уже сейчас — все классы-обертки над примитивными типами — неизменяемые. **Integer, Byte, Character, Short, Boolean, Long, Double, Float** — все эти классы создают *Immutable* объекты. Сюда же относятся и классы, используемые для создания больших чисел — **BigInteger** и **BigDecimal**.

Мы недавно проходили исключения и затрагивали **StackTrace**. Так вот: объекты класса **java.lang.StackTraceElement** тоже неизменяемые. Это логично: если бы кто-то мог изменять данные нашего стэка, это могло бы свести на нет всю работу с ним. Представьте, что кто-нибудь заходит в **StackTrace** и меняет **OutOfMemoryError** на **FileNotFoundException**. А вам с этим стеком работать и искать причину ошибки. А программа при этом вообще не использует файлы :) Поэтому от греха подальше эти объекты сделали неизменяемыми. Ну, со **StackTraceElement** более-менее понятно.

А зачем кому-то понадобилось делать неизменяемыми строки? В чем проблема, если бы можно было менять их значения. Наверное, даже удобнее бы было :/ Причин тут несколько. Во-первых, экономия памяти. Неизменяемые строки можно помещать в String Pool и использовать каждый раз одну и ту же вместо создания новых. Во-вторых, безопасность. Например, большинство логинов и паролей в любой программе — строки. Возможность их изменения могла бы повлечь проблемы с авторизацией.