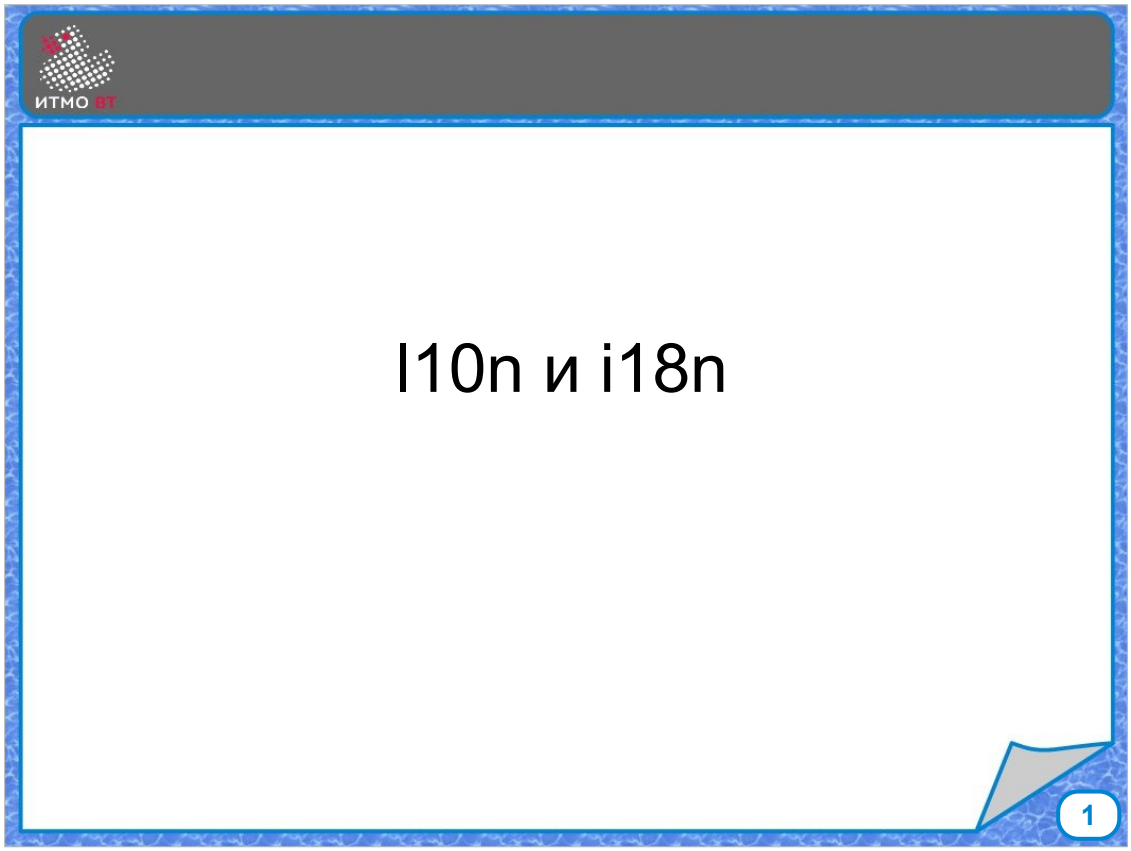
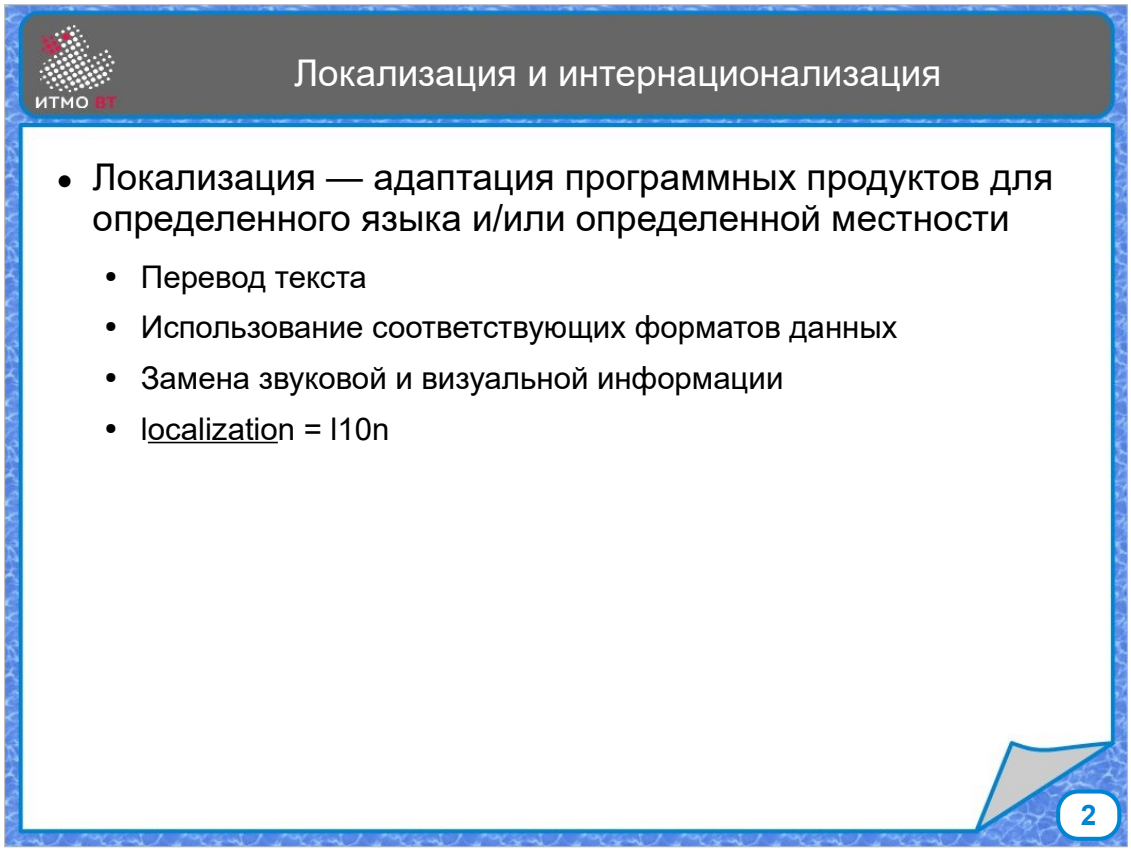
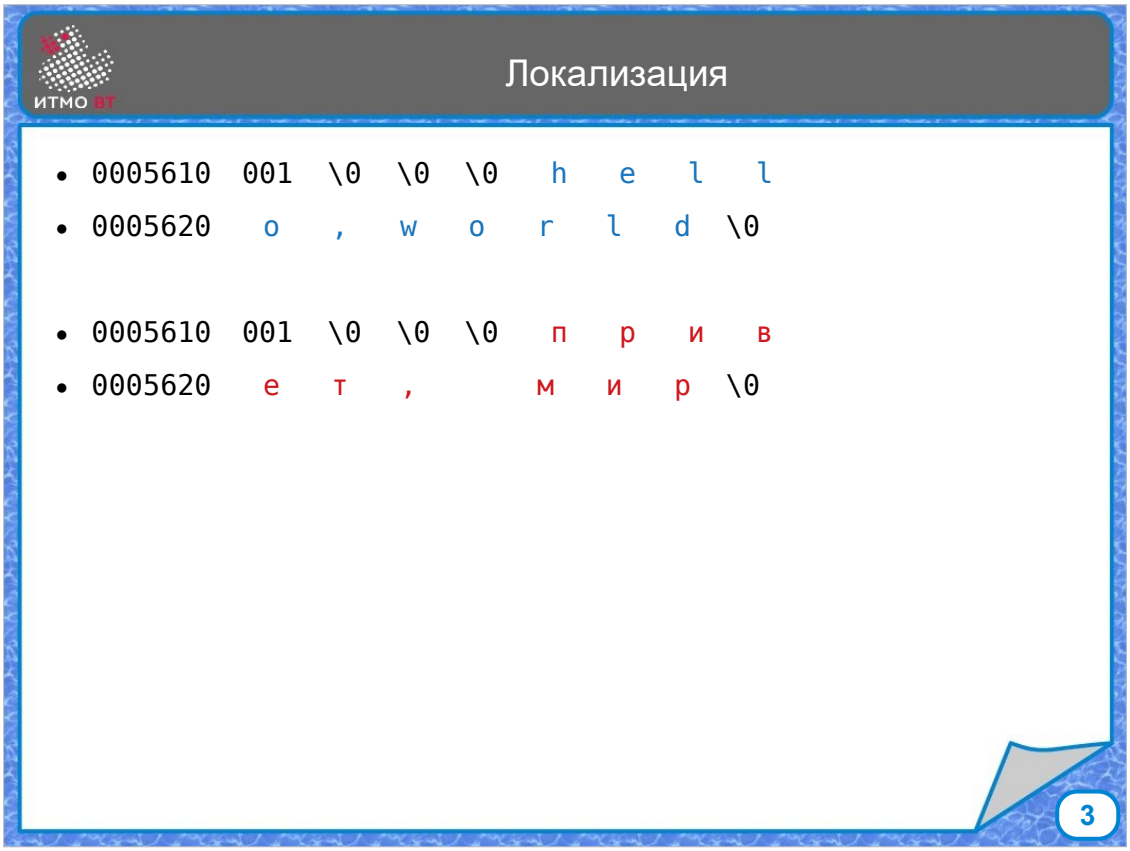
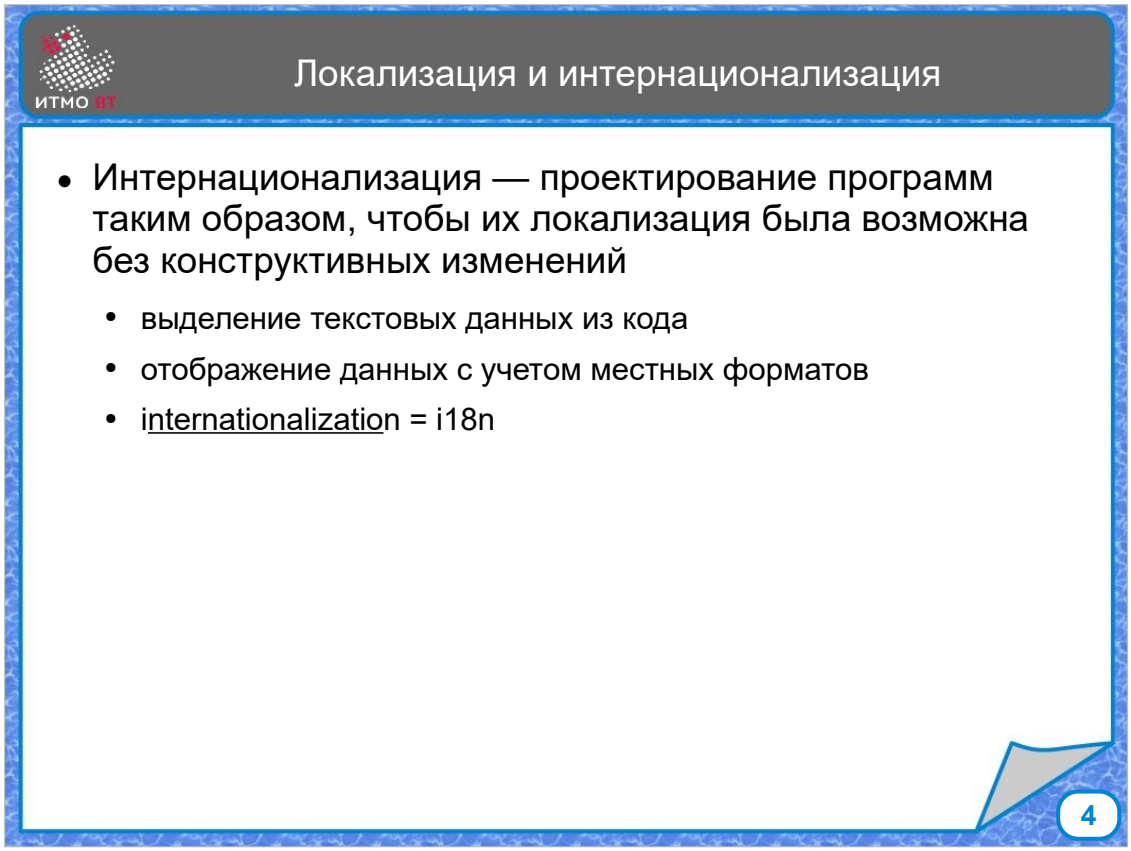

ИТМО ВТ
l10n и i18n
1



Локализация — адаптация программы для определенного языка или местности. Локализация обычно включает, перевод текста на другой язык, использование определенных форматов чисел, отображения даты и времени, замена звуковой и визуальной информации. Сокращенное обозначение локализации - l10n, слово localization начинается на l, заканчивается на n, и между ними еще 10 букв.

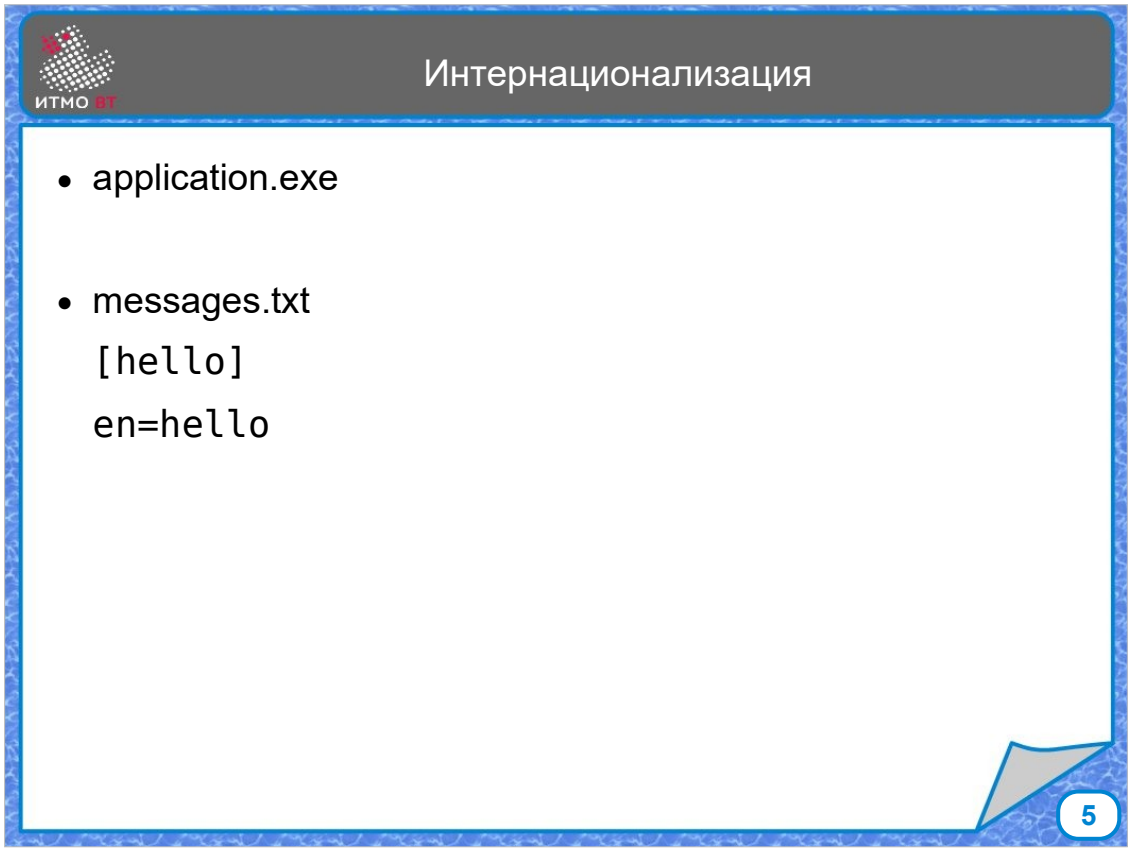


Простейший способ минимальной локализации - залезть в бинарник и поменять там нужную строчку на другую - так делали древние пираты софта в прошлом тысячелетии. Чуть более продвинутый способ — взять исходник, перевести строки, скомпилировать заново. Это все работает, но это не удобно, и не решает проблемы с форматами данных. При любом изменении исходной программы придется заново все менять.

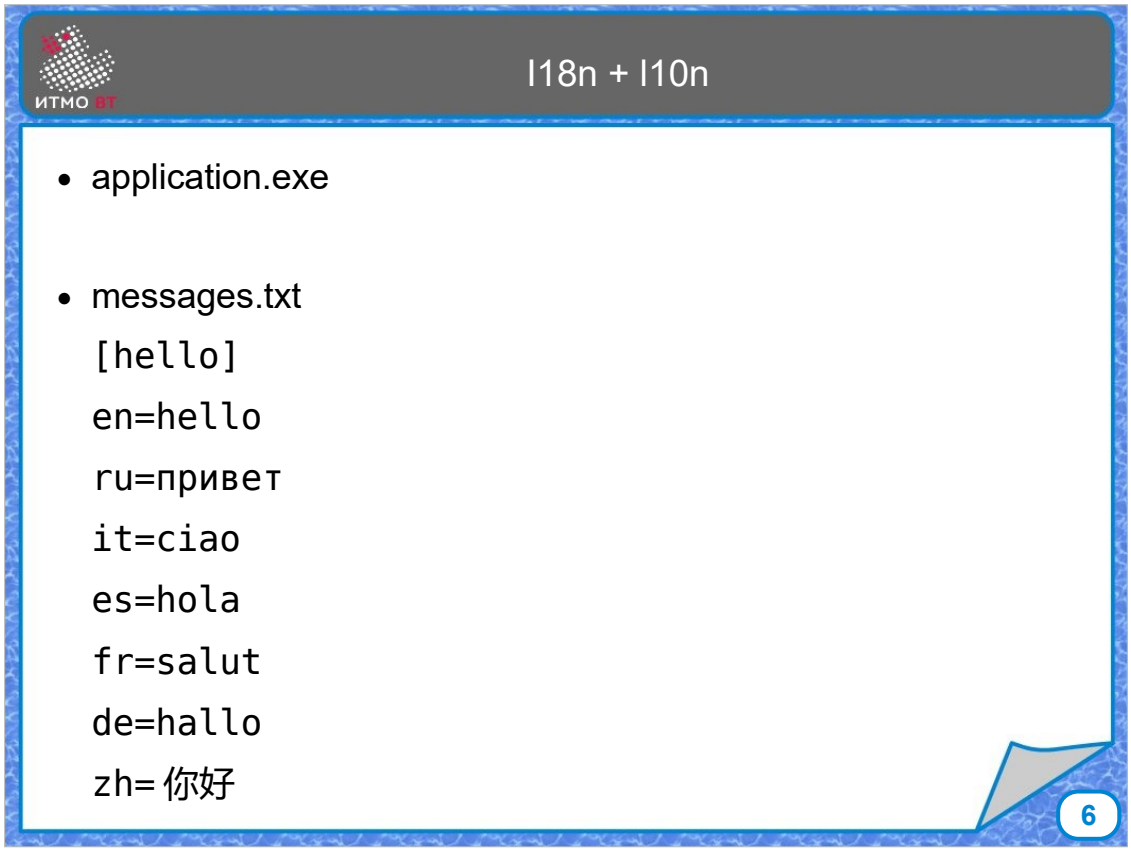


Для удобства выполнения локализации применяется интернационализация. Это не процесс, это стиль, как правильно писать программу, чтобы потом ее было удобно локализовывать.

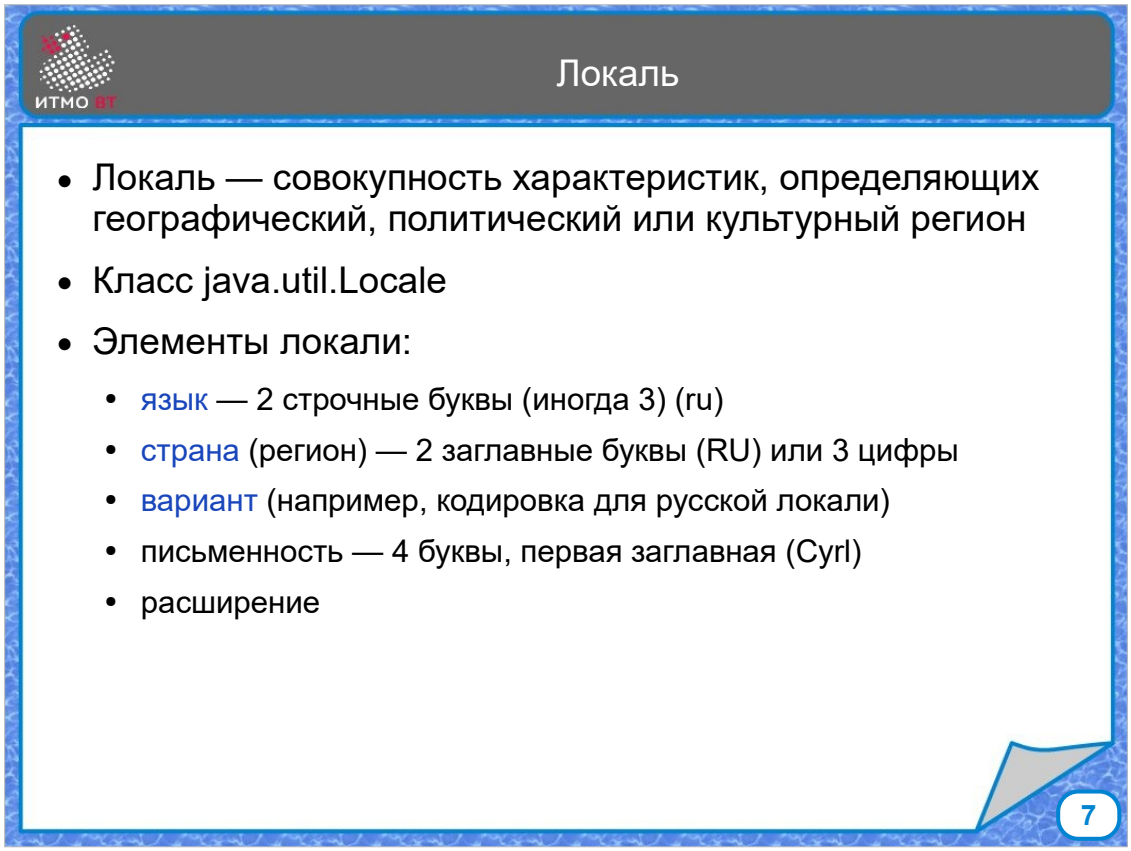
Интернационализация обычно заключается в том, что текстовые данные отделяются от кода и хранятся отдельно, а числа и даты отображаются не напрямую, а прогоняются через форматтеры, которые преобразуют данные в нужный формат. Сокращения слова интернационализация — это i18n, аналогично l10n.



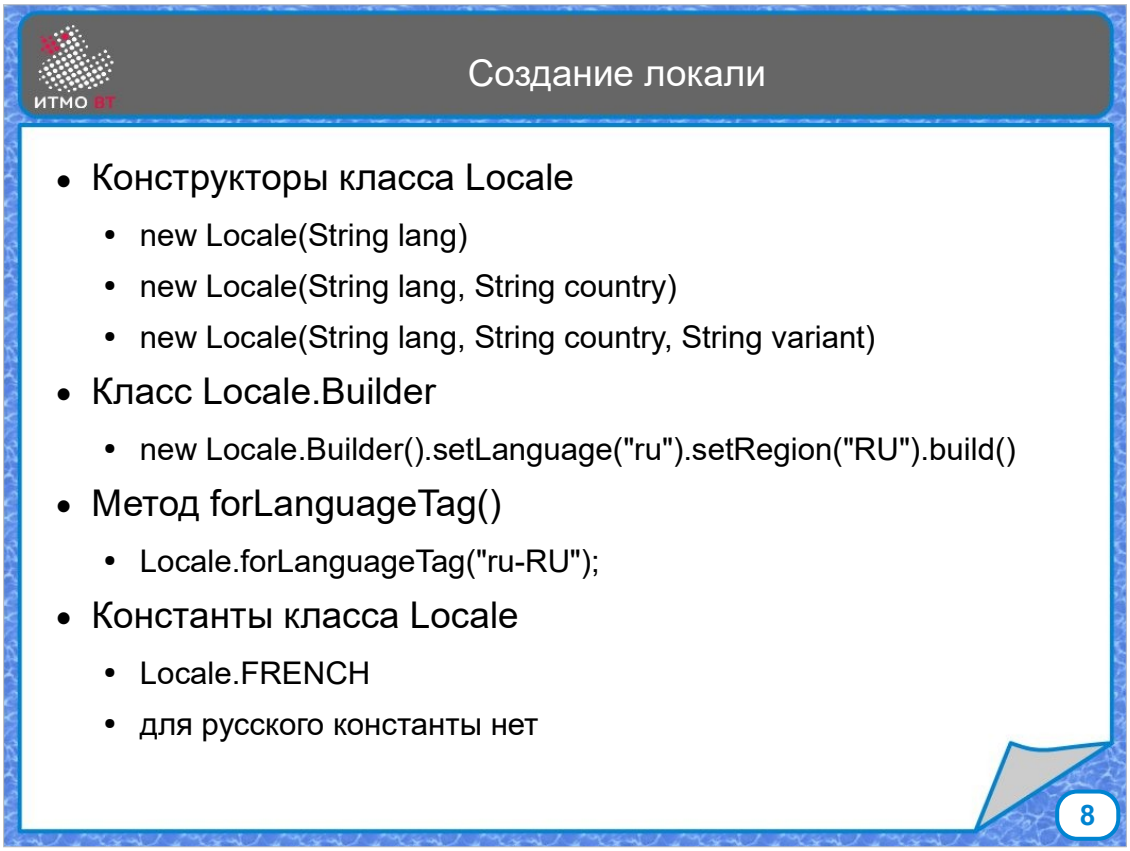
Есть у нас некое приложение, есть дополнительный текстовый файл, в нем есть некоторый раздел, и дальше мы пишем, что при использовании английского языка нужно будет вывести строку «hello». Это уже интернационализованное приложение, но оно пока не локализовано.



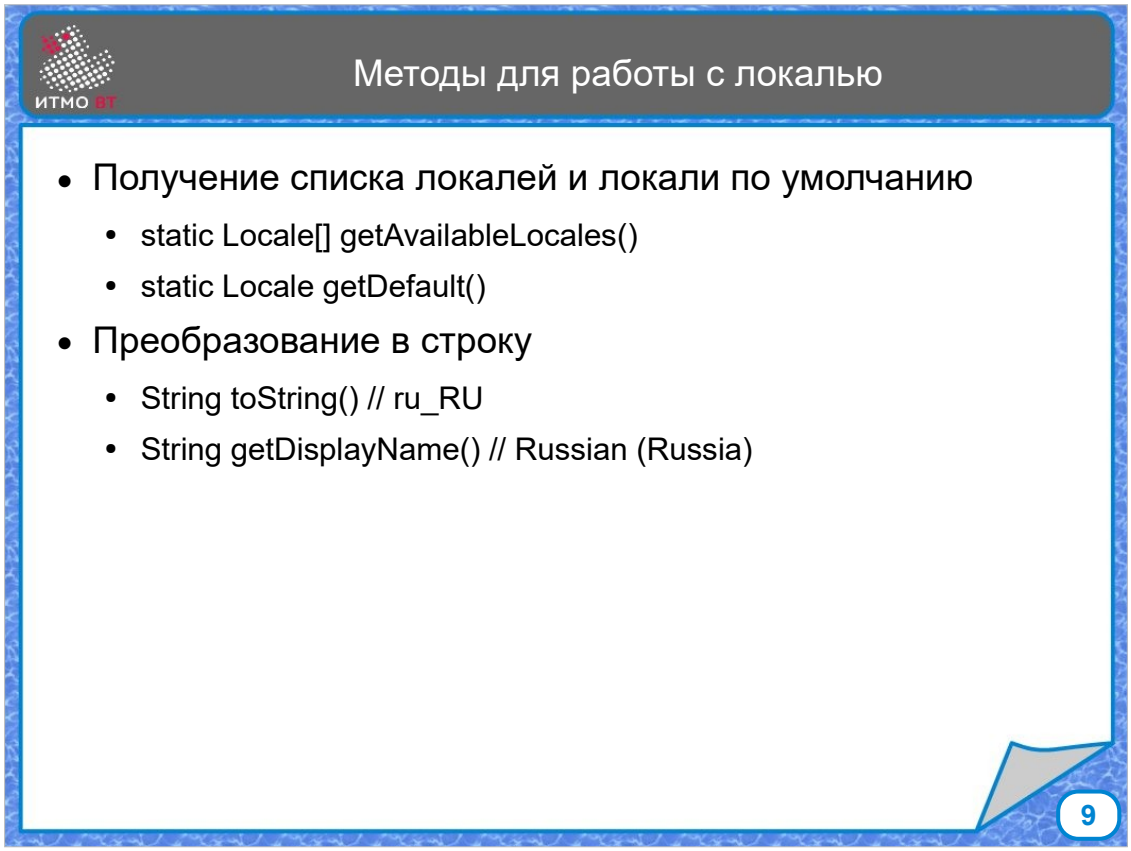
Чтобы оно стало локализованным, надо текстовый файл дополнить. Код уже трогать не надо. Остается только раздать текстовые файлы переводчикам, которые добавят поддержку языков. Это приложение уже интернационализованное и локализованное одновременно.



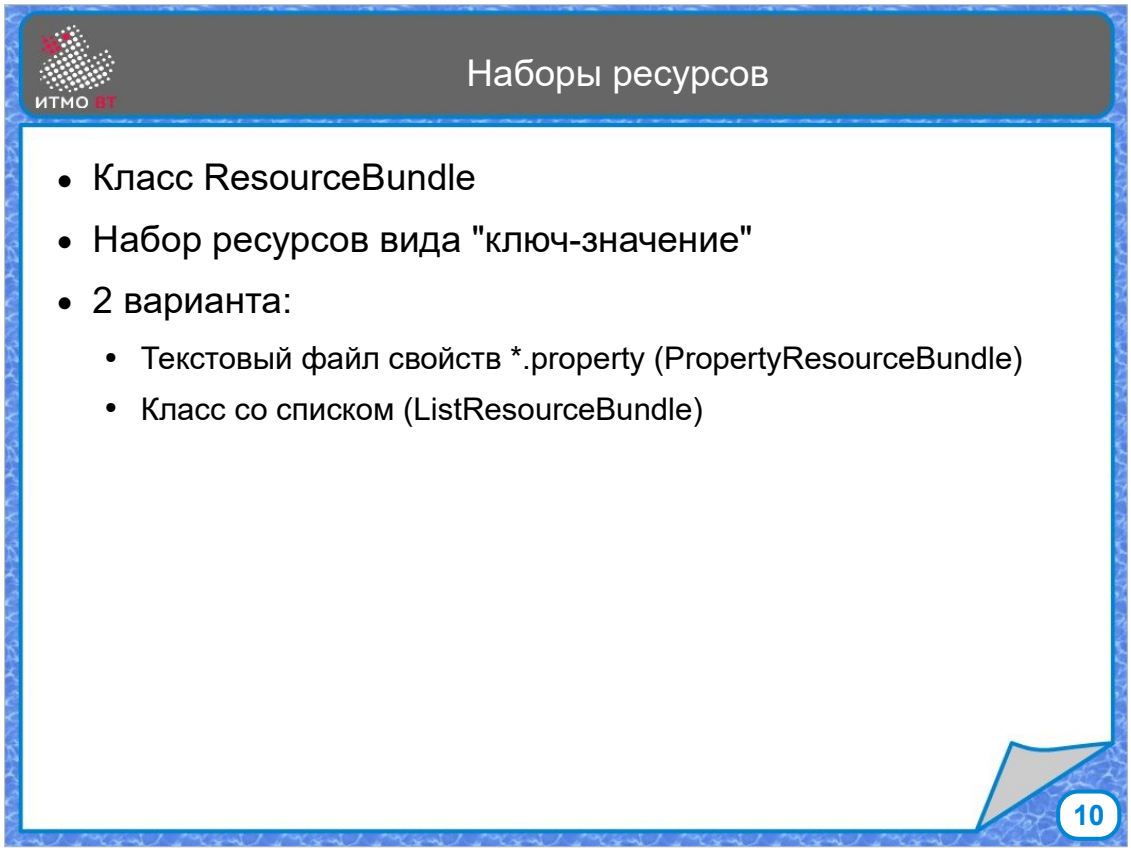
Локаль — это объект, задающий местность и язык. Элементы локали — это язык, страна и вариант, изредка встречаются дополниимельные элементы. Язык обозначается двумя строчными буквами: ru, en. Страна - обычно двумя заглавными буквами, иногда тремя цифрами. Вариант - показывает либо кодировку, либо применяемый календарь, если для данного языка и страны возможно несколько вариантов.



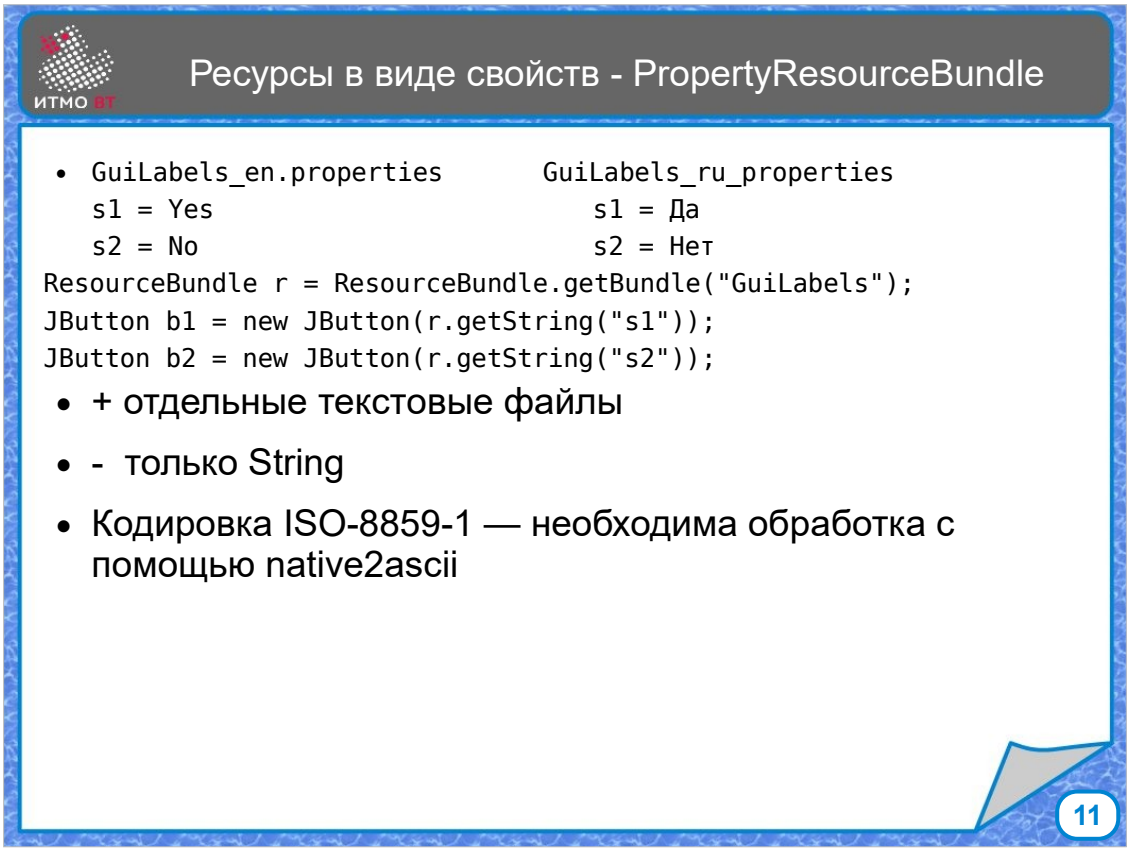
Создать локаль можно несколькими способами. Можно использовать конструктор, и создать локаль из языка и страны. Сама по себе локаль не содержит в себе никаких данных, о применяемых для данной локали форматах, валютах и т. д., локаль - это просто обозначение. Информация о форматах и других параметрах хранится в отдельных файлах или классах. Другим способом получить локаль является класс LocaleBuilder. И, кроме этого, можно использовать либо статический метод Locale.forLanguageTag, либо для некоторых локалей есть константы.



Есть возможность получить список всех локалей, которые доступны в данный момент. Можно получить локаль по умолчанию, который сейчас стоит в системе. Кстати, многие методы, которые позволяют выполнять зависящие от локали действия, используют дефолтную локаль, если она явно не задана в методе.

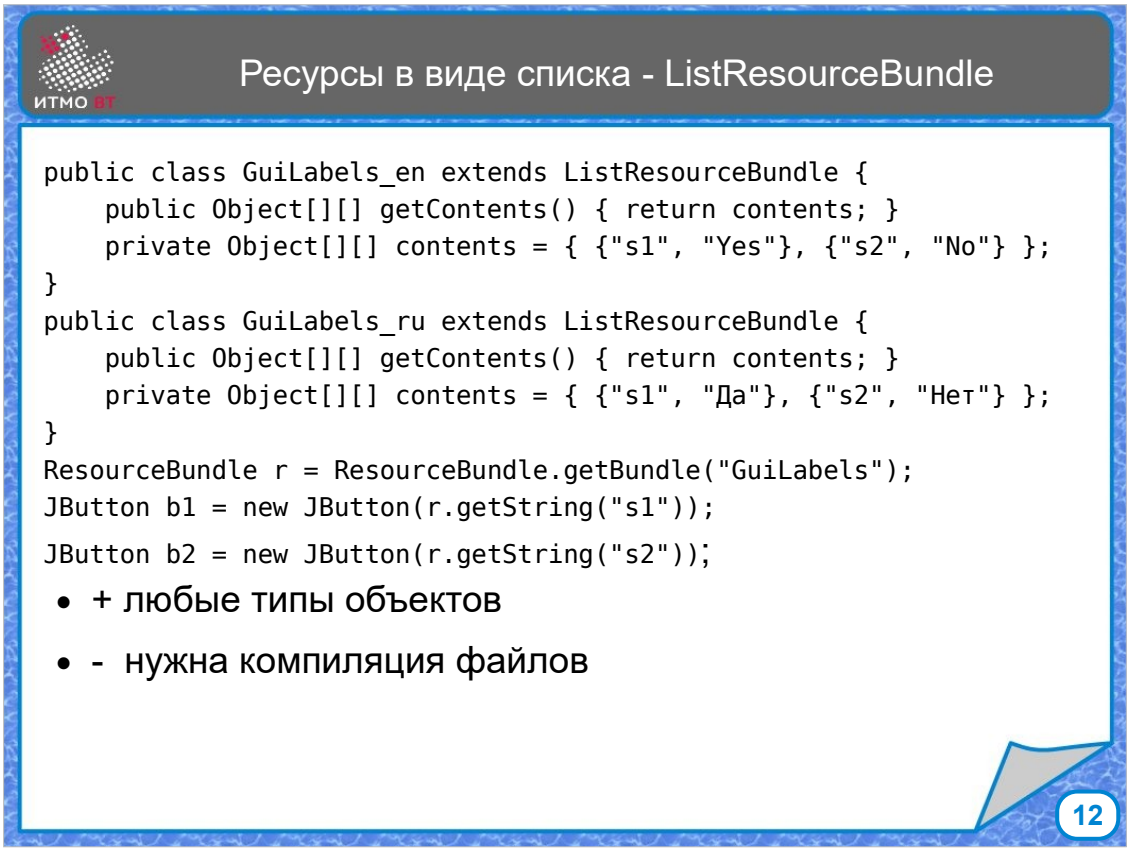


Тексты, выделенные для локализации, хранятся в виде пар ключ-значение. Для каждого ключа хранится множество значений, соответствующих локалям. Ими можно управлять с помощью класса ResourceBundle. Это абстрактный класс, у которого есть два потомка - PropertyResourceBundle и ListResourceBundle. Первый используется для хранения исключительно текствой информации (при этом его проще использовать), второй можно применять для хранения любых объектов.

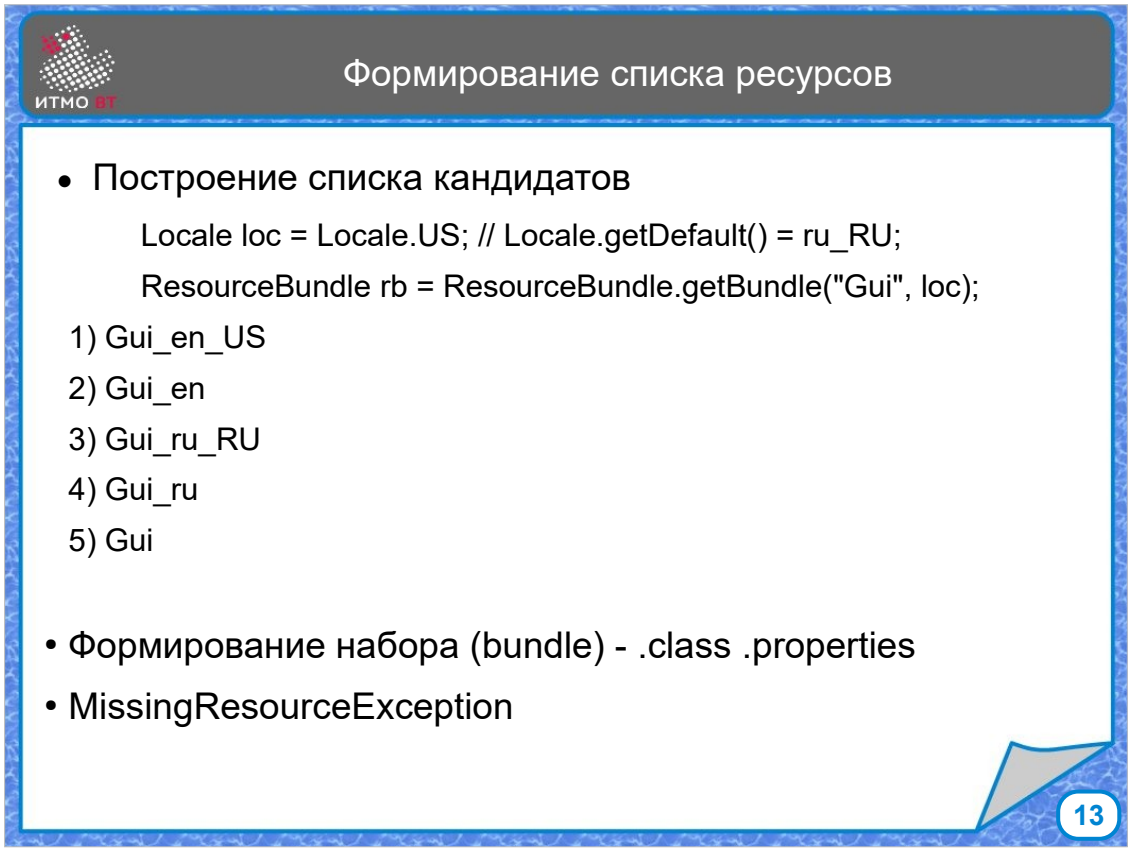


При использовании PropertyResourceBundle достаточно создать текстовые файлы с опредленным именем. Имя должно состоять из названия ресурса, локали и расширения .properties. Несколько файлов с одним именем ресурса, но разными локалями образуют набор ресурсов (ResourceBundle).

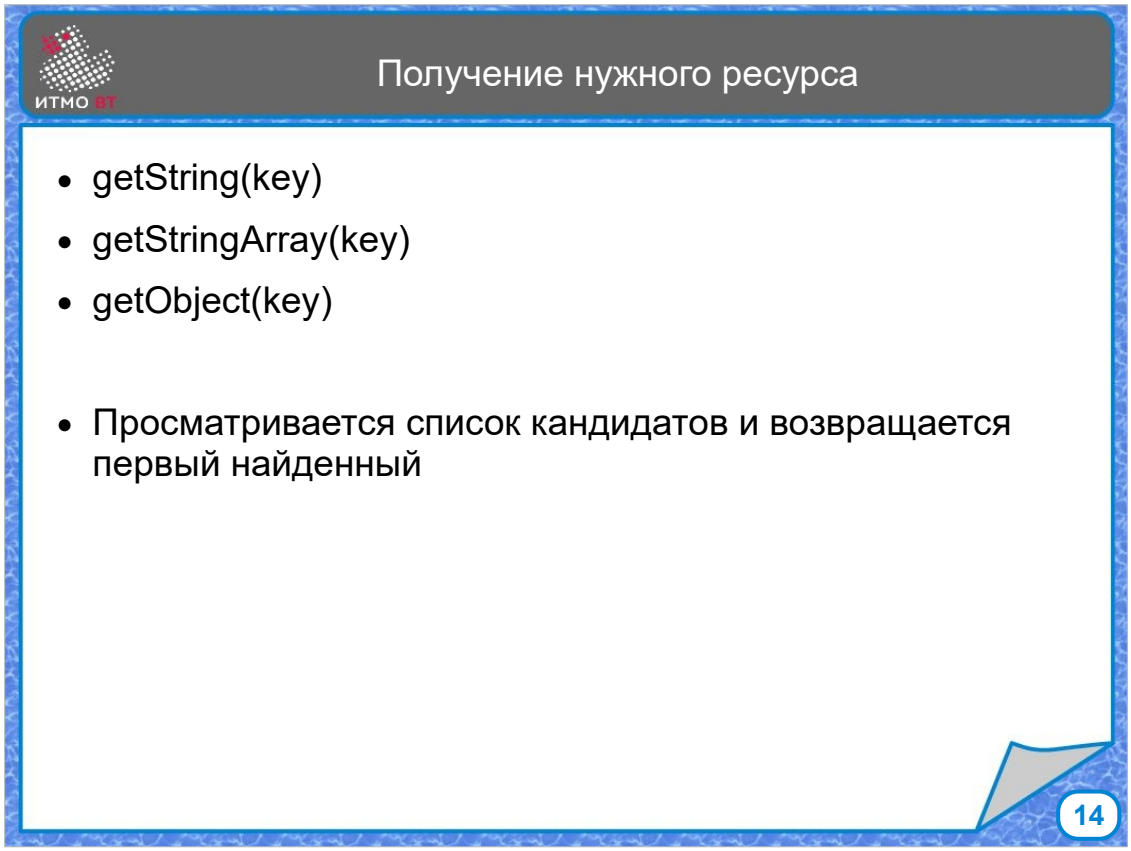
С помощью метода getBundle набор ресурсов загружается и после этого его можно использовать. Текстовые данные берутся из набора ресурсов по ключу, возможно с указанием локали. Основное преимущество файлов свойств - простой формат файлов с данными. Недостаток - возможность использовать только текстовые данные. Особенность в том, что кодировка у файлов свойств должна быть ISO 8859-1, т. е. это латиница. Для перевода файлов из других кодировок можно использовать утилиту native2ascii, входящую в комплект JDK.



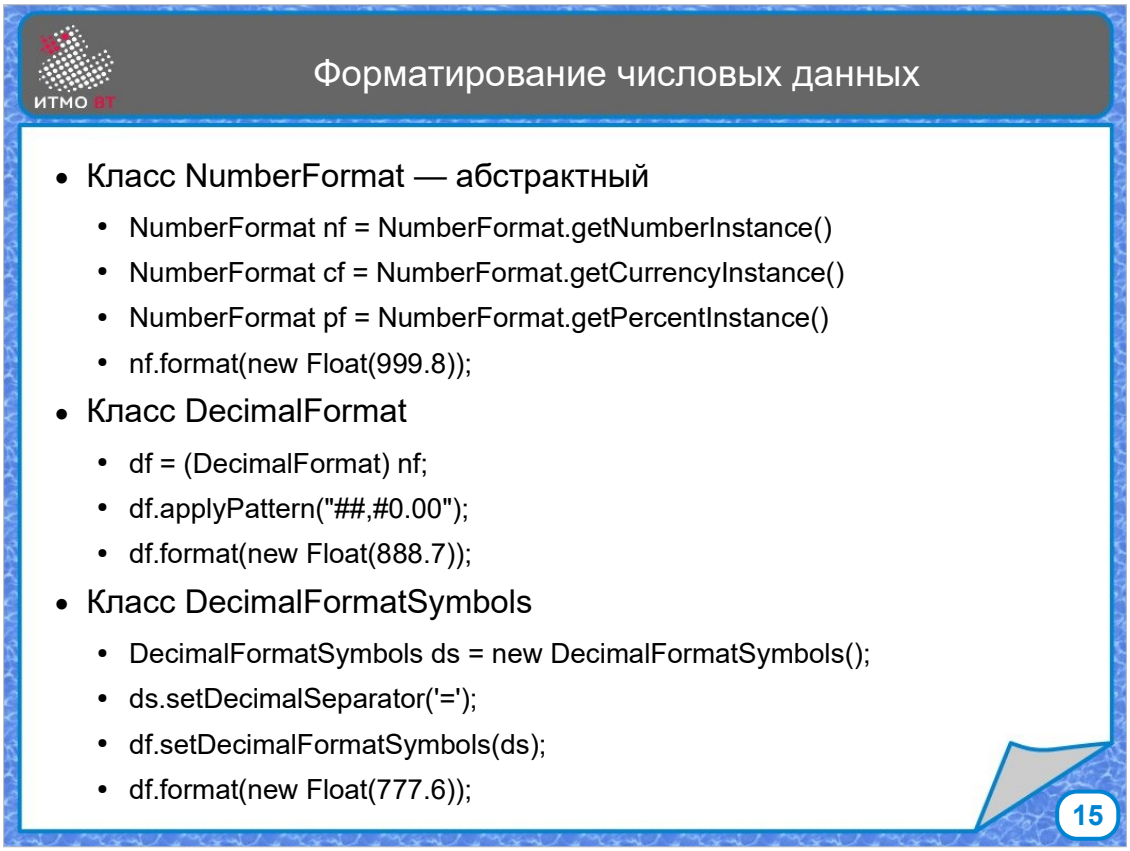
Если пользоваться классом ListResourceBundle, то будет немного другое решение. Нужно создать наследников класса ListResourceBundle с именами этих классов, сформированными по тому же принципу как и у файлов свойств, Затем в каждом классе реализовать метод getContents(), возвращающий двумерный массив объектов, содержащий ключи и соответствующие им значения.
Такой способ немного сложнее, потому что формат класса более сложный, после того, как класс создан, его необходимо скомпилировать. Но при этом можно хранить любые объекты, не только текстовые.



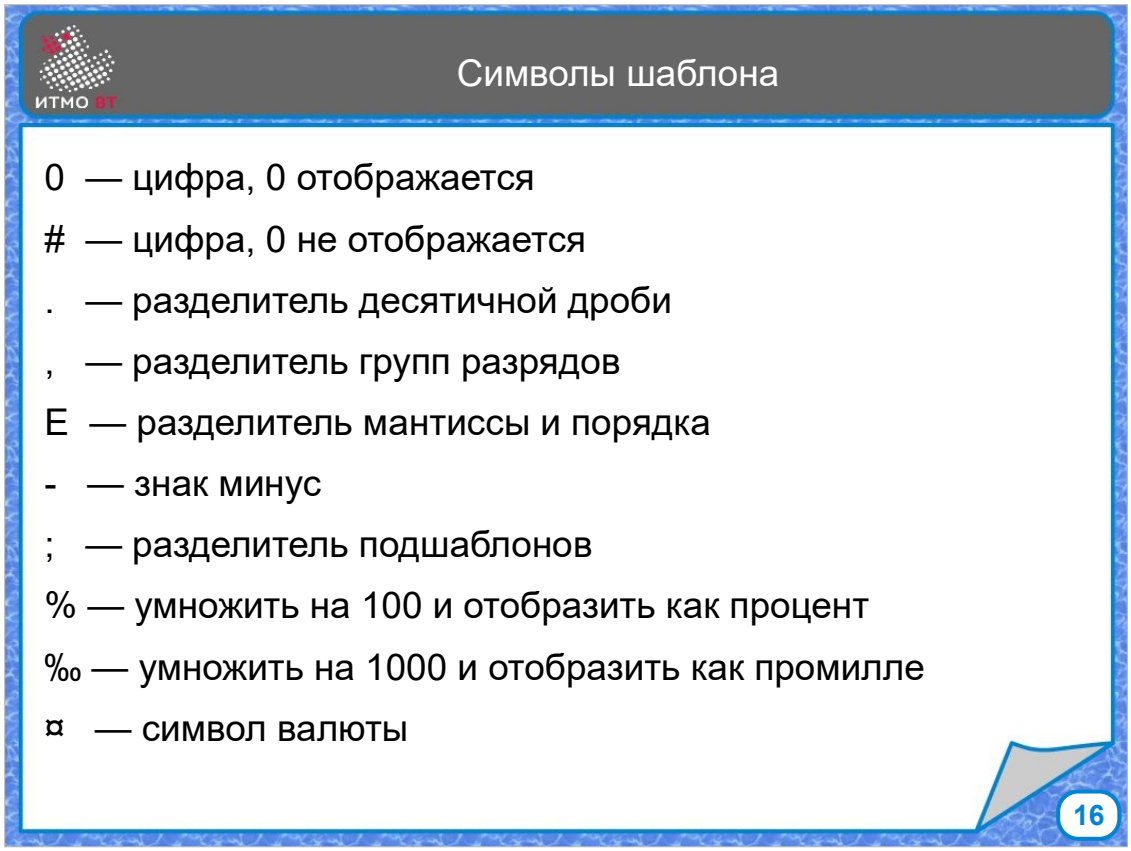
Принцип формировария списка кандидатов и поиска ресурсов. Методу getBundle передается название ресурса ("Gui") и, возможно, локаль. Первым будет ресурс с переданной локалью, если она указана. Ищем класс Gui_en_US.class. Если он есть, добавляем в список. Если нет, то ищем файл Gui_en_US.properties. Потом пробуем найти такие же ресурсы, но без указания страны, то есть класс или свойство с именем Gui_en. Все, что нашли, добавляем в набор. Повторяем те же действия для системной локали по умолчанию, то есть для имен Gui_ru_RU и Gui_ru. Последним в наборе будет ресурс без указания локали - Gui.class или Gui.properties. Если не нашлось ни одного класса или файла свойств с нужными именами, то выбрасывается исключение. В итоге в объекте класса ResourceBundle будет сформирован список классов и свойств в определенном порядке.



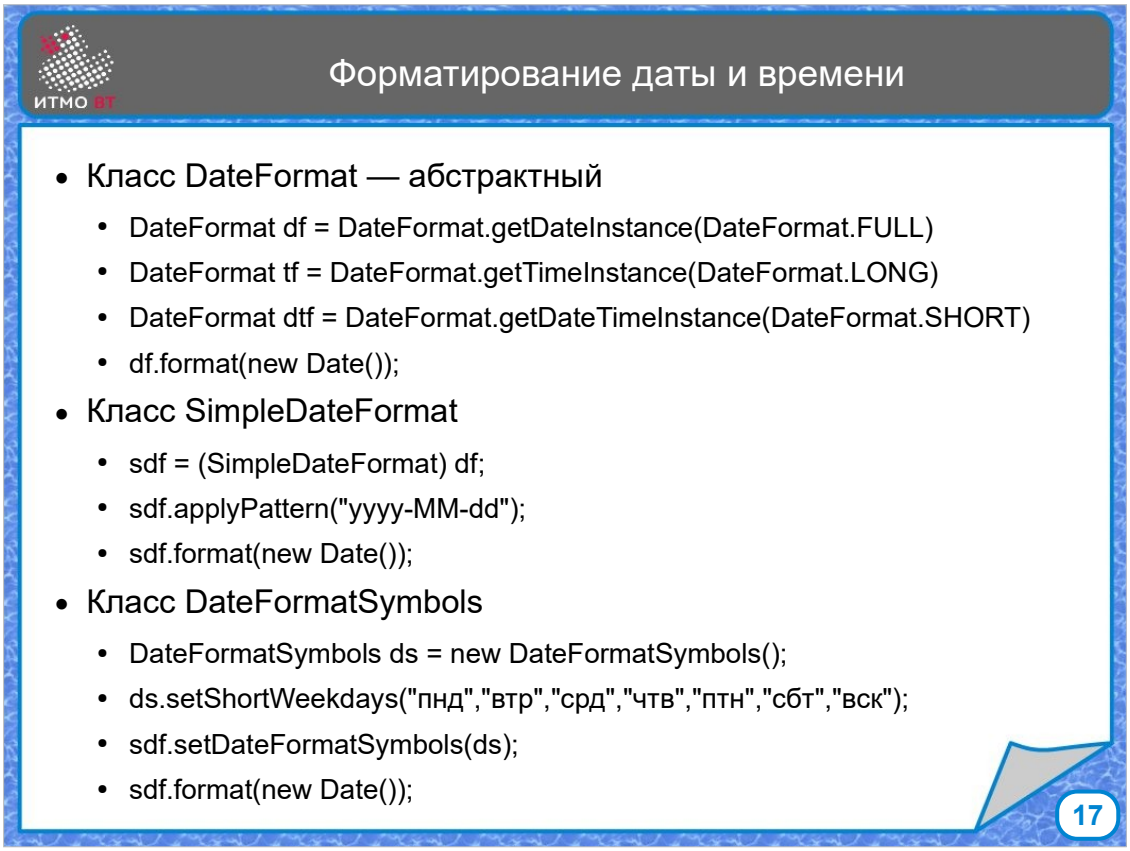
Теперь в какой-то момент нам нужно найти строку по ключу с помощью метода getString(), getStringArray или getObject. Поиск ведется в сформированном списке ресурсов последовательно. Как только ключ найден в одном из них, возвращается соответствующее ключу значение. В итоге получается так - сначала пытаемся найти нужный ресурс для точно соответствующей заданной локали (США, английский язык). Если его нет, то ищем просто для английского языка. Если и такого нет, то пробуем системную локаль (Россия, русский). Не нашли, пробуем просто русский язык. Если и его нет, тогда запасной вариант - ресурс без локали. Если и его не удалось найти - MissingResourceException.



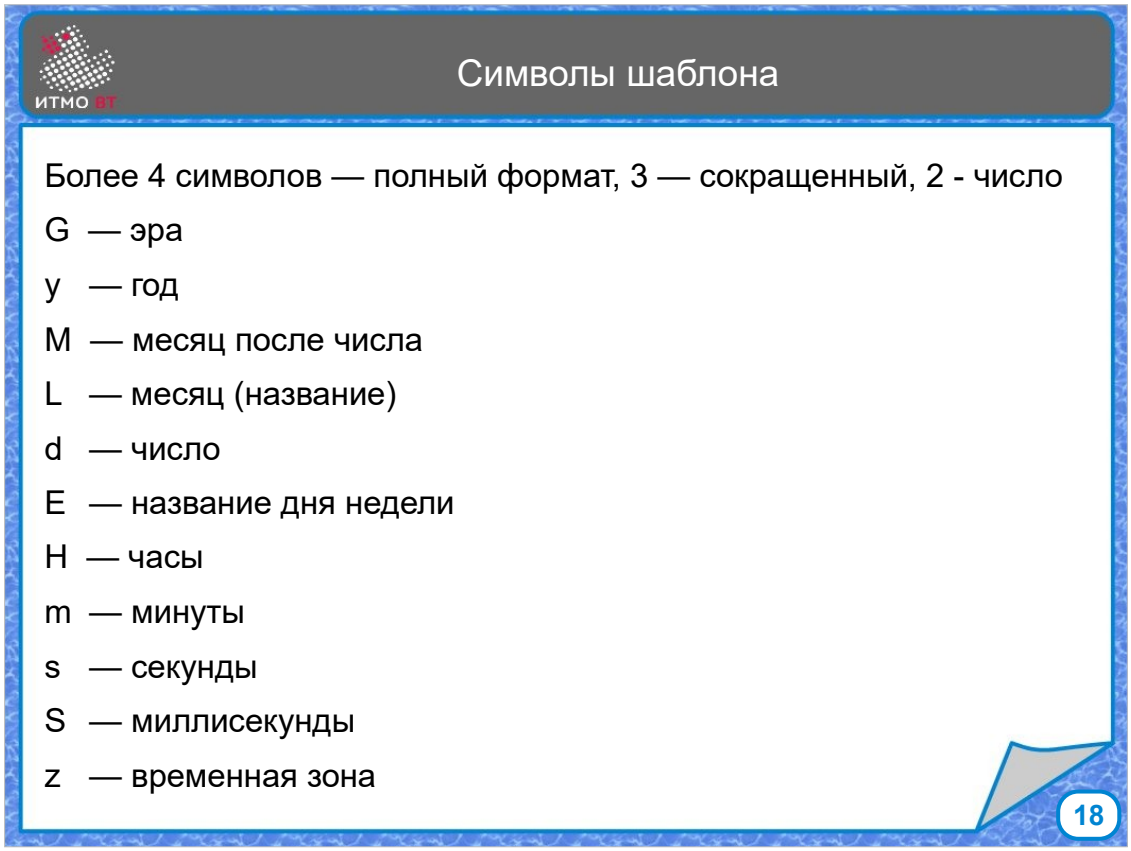
Для форматирования чисел используются следующие классы. NumberFormat - абстрактный класс с фабричными методами, которые возвращают одного из потомков NumberFormat, умеющего форматировать числа в соответствии с заданными правилами (для числа, для валюты и для процентов). Обычно этим потомком является объект класса DecimalFormat. С помощью метода format можно получить число в нужном формате. Наиболее общий формат для всех локалей выдают фабричные методы NumberFormat. Если нужен формат с другими параметрами (количество дробных знаков, наличие знака, и т.д.), можно самостоятельно создать объект DecimalFormat и в нем настроить формат. Класс DecimalFormatSymbols позволяет заменить в формате символы-разделители, если возникла необходимость это сделать.



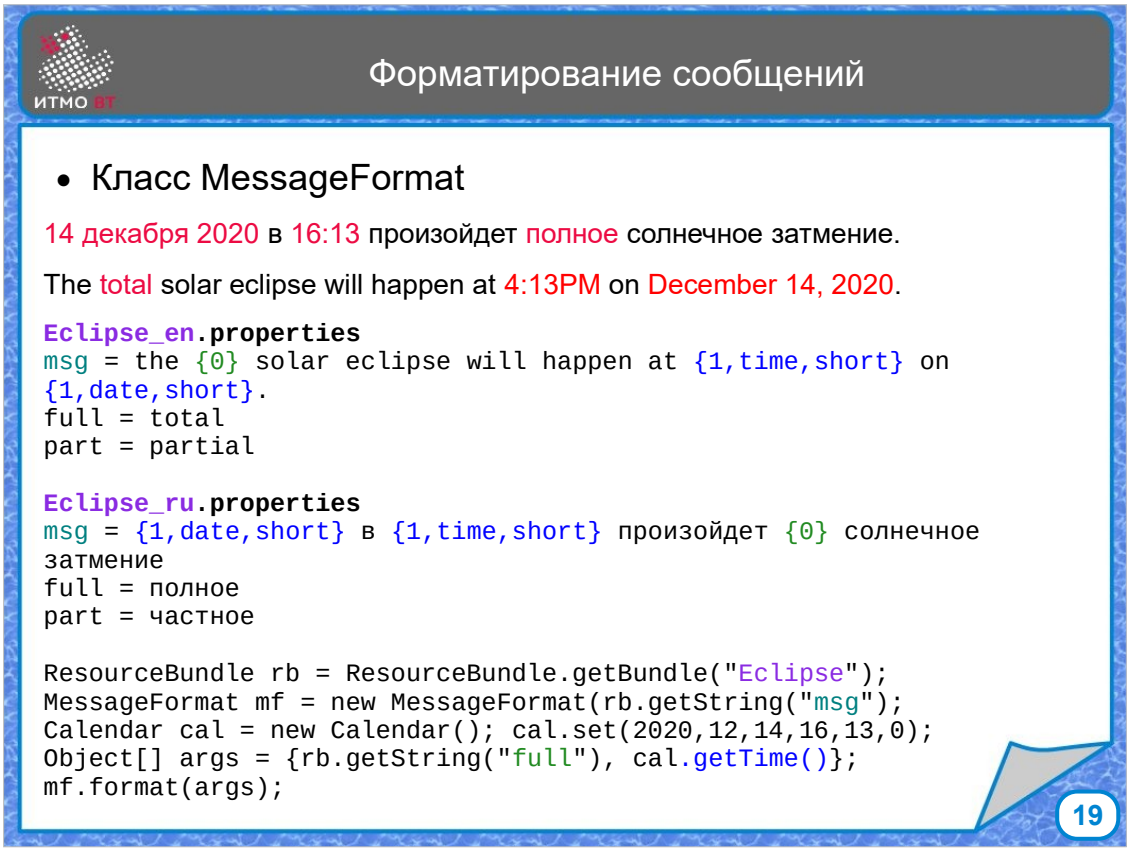
На этом слайде приведены символы шаблона, использующиеся для создания собственного формата, если не устраивает стандартный.



Похожая схема применяется для форматирования даты и времени. Абстрактный класс DateFormat имеет фабричные методы для получения форматтера для даты, времени, и совмещенного. Можно задать тип формата: полный, длинный, средний, короткий. Если локаль не задана, используется системная. Данные методы возвращают объект класса SimpleDateFormat, являющийся потомком DateFormat. SimpleDateFormat позволяет указать произвольный шаблон формата. И класс DateFormatSymbols используется для замены стандартных элементов формата даты или времени.



На этом слайде приведены символы шаблона, использующиеся для создания собственного формата, если не устраивает стандартный. Тип формата (полный, сокращенный, числовой) задается повтором символов определенное число раз. Например, MMMM - это месяц в полном формате.



Для поддержки разного порядка слов в разных языках есть класс MessageFormat, который позволяет использовать параметры в строке. Соответственно, можно задать для разных языков строки с параметрами в разных местах. На слайде показан пример для фраз на русском и английском языках, где в строку подставляется время, дата и другая строка.

Формат с выбором

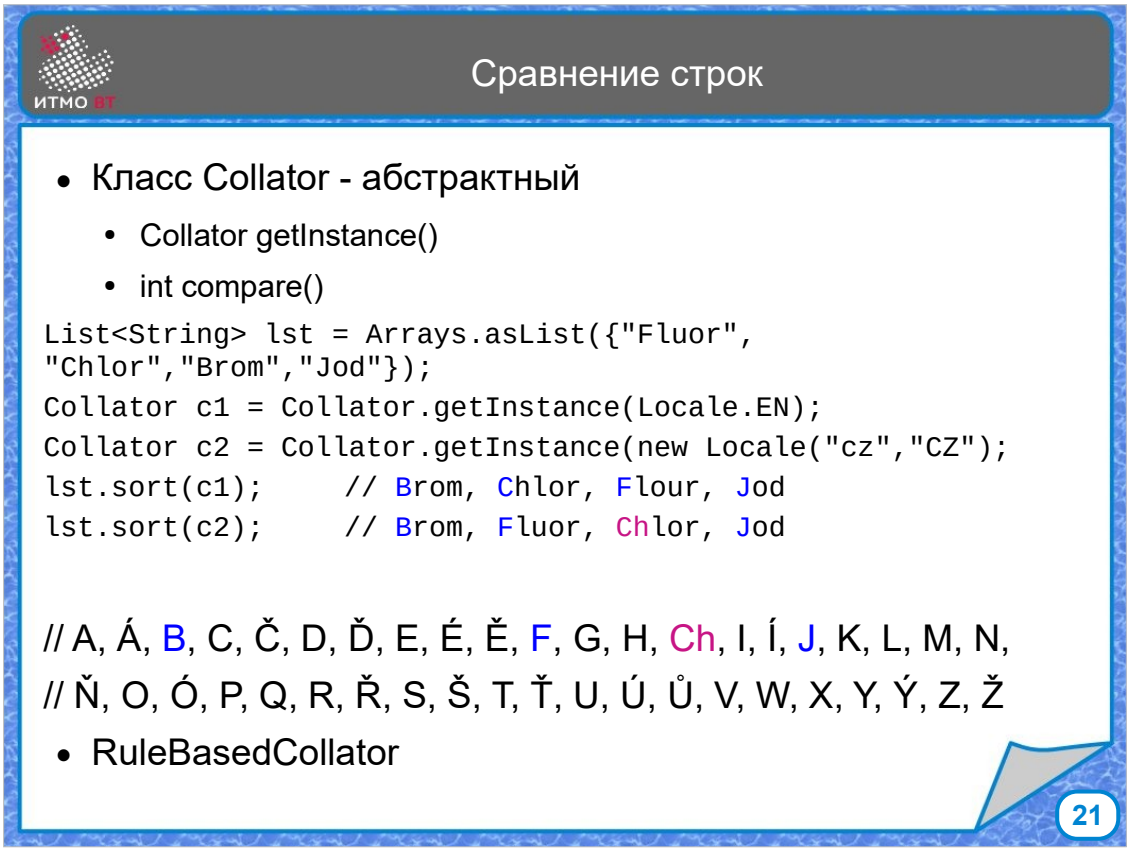
- Класс ChoiceFormat extends NumberFormat
  - 0 friends like it
  - 1 friend likes it
  - 1000 friends like it

Like_en.properties

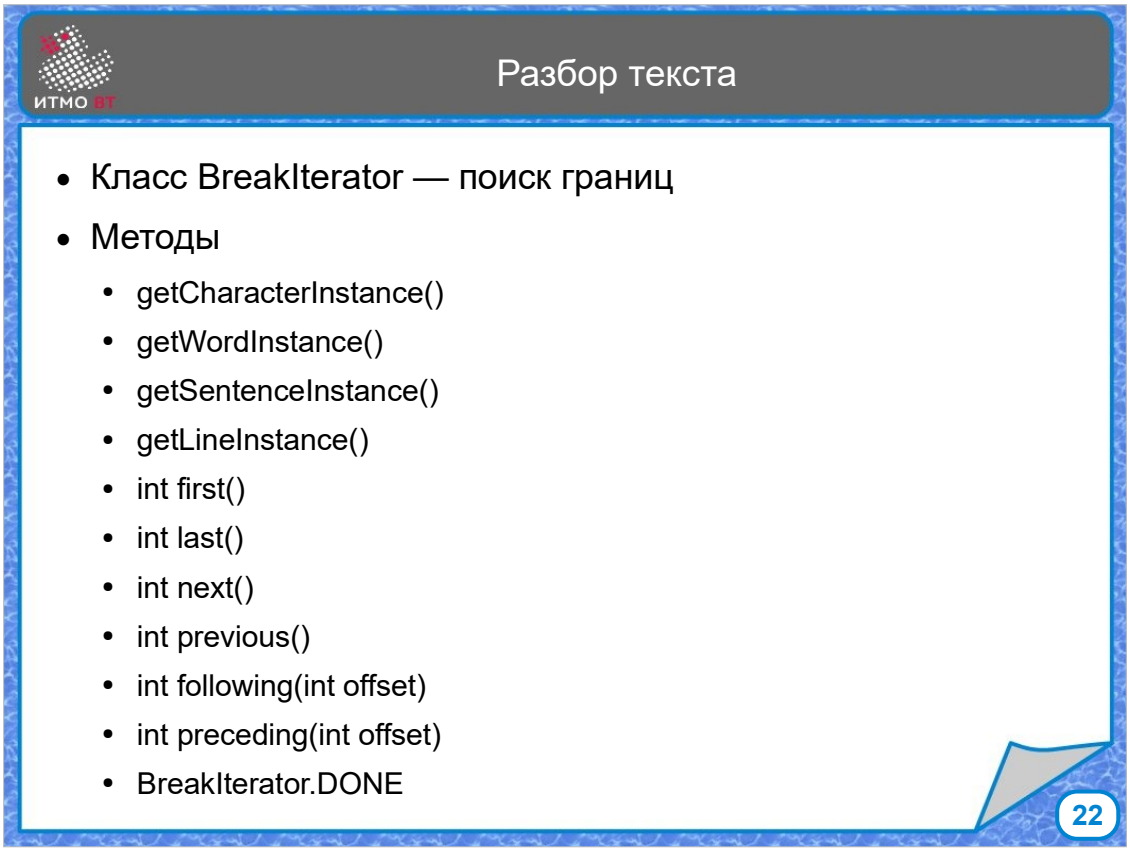
```
msg = {0} it
one = {0,number} friend likes
many = {0,number} friends like
```

```
ResourceBundle rb = ResourceBundle.getBundle("Like");
MessageFormat mf = new MessageFormat(r.getString("msg"));
String one = rb.getString("one");
String many = rb.getString("many");
double[] lims = { 0, 1, 2 };
String[] msgs = { many, one, many };
ChoiceFormat cf = new ChoiceFormat(lims, msgs);
mf.setFormatByArgumentIndex(0, cf);
Object[] args = { new Integer(15) };
mf.format(args);
```


Класс ChoiceFormat позволяет выбрать и подставитьт один вариант из нескольких в зависимости от используемого значения. Его можно использовать, например, для выбора нужной формы слова при числительных (хотя для русского языка это реализовать сложнее, чем для английского). Для создания формата нужно задать массив чисел, задающих диапазоны, и соответствующие этим диапазонам ключи.



Для сравнения строк в разных языках используется класс Collator. У него есть методы getInstance и compare. Метод getInstance возвращает экземпляр класса RuleBasedCollator для нужной локали. Он умеет обрабатывать случаи, когда в языке буква обозначается двумя символами. Класс Collator имеет 2 свойства - режим декомпозиции, задающий порядок сопоставления простых и составных символов Unicode (NO, CANONICAL и FULL), и сила, которая определяет, какие именно характеристики символов (PRIMARY, SECONDARY, TERTIARY, IDENTICAL) учитываются для признания их разными или одинаковыми. Это позволяет настраивать сравнение, например считать ли эквивалентными 'a' и 'A', либо 'a' и 'á'



Ну и, наконец, класс BreakIterator позволяет разбивать текст на составляющие - строки, предложения, слова, символы, в зависимости от установленной локали. С его помощью можно получить нужный итератор для разных случаев.