В первом случае к первой группе относятся все возможные символы, но при этом остается минимальное количество символов для второй группы.

Во втором случае для первой группы выбирается наименьшее количество символов, т. к. используется слабое совпадение.

В третьем случае первой группе будет соответствовать вся строка, а для второй не остается ни одного символа, так как вторая группа использует слабое совпадение.

В четвертом случае строка не соответствует регулярному выражению, т. к. для двух групп выбирается наименьшее количество символов.

В классе **Matcher** объявлены два полезных метода для замены найденных подпоследовательностей во входной строке.

Matcher appendReplacement(StringBuffer sb, String replacement) – метод читает символы из входной стоки и добавляет их в **sb**. Чтение останавливается на **start()-1** позиции предыдущего совпадения, после чего происходит добавление в **sb** строки **replacement**. При следующем вызове этого метода производится добавление символов, начиная с символа с индексом **end()** предыдущего совпадения.

StringBuffer appendTail(StringBuffer sb) – добавляет оставшуюся часть символов из входной последовательности в **sb**. Как правило, вызывается после одного или нескольких вызовов метода **appendReplacement()**.

Интернационализация текста

Класс **java.util.Locale** позволяет учесть особенности региональных представлений алфавита, символов и проч. Автоматически виртуальная машина использует текущие региональные установки операционной системы, но при необходимости их можно изменять. Для некоторых стран региональные параметры устанавливаются с помощью констант, например: **Locale.US**, **Locale.FRANCE**. Для других стран объект **Locale** нужно создавать с помощью конструктора:

```
Locale myLocale = new Locale("bel", "BY");
```

Получить доступ к текущему варианту региональных параметров можно следующим образом:

```
Locale current = Locale.getDefault();
```

Если, например, в ОС установлен регион «Россия» или в приложении с помощью **new Locale("ru", "RU")**, то следующий код (при выводе результатов выполнения на консоль)

```
current.getCountry();//код региона
current.getDisplayCountry();//название региона
current.getLanguage();//код языка региона
current.getDisplayLanguage();//название языка региона
```

позволяет получить информацию о регионе в виде:

```
RU
Россия
ru
русский
```

Для создания приложений, поддерживающих несколько языков, существует целый ряд решений. Самое логичное из них – использование взаимодействия классов **java.util.ResourceBundle** и **Locale**. Класс **ResourceBundle** предназначен в первую очередь для работы с текстовыми файлами свойств (расширение **.properties**). Каждый объект **ResourceBundle** представляет собой набор объектов соответствующих подтипов, которые разделяют одно и то же базовое имя, к которому можно получить доступ через поле **parent**. Следующий список показывает возможный набор соответствующих ресурсов с базовым именем **text**. Символы, следующие за базовым именем, показывают код языка, код страны и тип операционной системы. Например, файл **text_de_CH.properties** соответствует объекту **Locale**, заданному кодом языка немецкого (**de**) и кодом страны Швейцарии (**CH**).

```
text.properties
text_ru.properties
text_de_CH.properties
text_en_CA_UNIX.properties
```

Чтобы выбрать определенный объект **ResourceBundle**, следует вызвать метод **ResourceBundle.getBundle(**параметры**)**. Следующий фрагмент выбирает **text** объекта **ResourceBundle** для объекта **Locale**, который соответствует английскому языку, стране Канаде и платформе UNIX.

```
Locale currentLocale = new Locale("en", "CA", "UNIX");
ResourceBundle rb =
   ResourceBundle.getBundle("text", currentLocale);
```

Если объект **ResourceBundle** для заданного объекта **Locale** не существует, то метод **getBundle()** извлечет наиболее общий. В случае если общее определение файла ресурсов не задано, то метод **getBundle()** генерирует исключительную ситуацию **MissingResourceException**. Чтобы этого не произошло, необходимо обеспечить наличие базового файла ресурсов без суффиксов, а именно: **text.properties** в отличие от частных случаев вида:

```
text_en_US.properties
text_bel_BY.properties
```

В файлах свойств информация должна быть организована по принципу:

```
key1 = value1
key2 = value2
...
```

Например: **str1 = To be or not to be?**

Перечисление всех ключей в виде **Enumeration<String>** можно получить вызовом метода **getKeys()**. Конкретное значение по ключу извлекается методом **String getString(String key)**.

В следующем примере в зависимости от выбора пользователя известная фраза будет выведена на одном из трех языков.

// пример # 16 : поддержка различных языков: HamletInternational.java

```
package chapt8;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.Locale;
```

```
import java.util.ResourceBundle;

public class HamletInternational {
      public static void main(String[] args) {
            String country = "", language = "";
            System.out.println("1 - Английский");
            System.out.println("2 - Белорусский");
            System.out.println("Любой символ - Русский");
            char i = 0;
            try {
                  i = (char) System.in.read();
            } catch (IOException e1) {
                  e1.printStackTrace();
            }
            switch (i) {
             case '1':
                  country = "US";
                  language = "EN";
                  break;
             case '2':
                  country = "BY";
                  language = "BEL";
            }
            Locale current = new Locale(language, country);
            ResourceBundle rb =
            ResourceBundle.getBundle("text", current);
            try {
                  String st = rb.getString("str1");
                  String s1 =
            new String(st.getBytes("ISO-8859-1"), "UTF-8");
                  System.out.println(s1);

                  st = rb.getString("str2");
                  String s2 =
            new String(st.getBytes("ISO-8859-1"), "UTF-8");
                  System.out.println(s2);
            } catch (UnsupportedEncodingException e) {
                  e.printStackTrace();
            }
      }
}
```

Файл **text_en_US.properties** содержит следующую информацию:

```
str1 = To be or not to be?
str2 = This is a question.
```

Файл **text_bel_BY.properties**:

```
str1 = Быць або не быць?
str2 = Вось у чым пытанне.
```

Файл **text.properties**:

```
str1 = Быть или не быть?
str2 = Вот в чём вопрос.
```