ЛАБОРАТОРНАЯ РАБОТА №12 ОБРАБОТКА СТРОК. ИСПОЛЬЗОВАНИЕ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ В JAVA-ПРИЛОЖЕНИЯХ

ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Цель данной лабораторной работы – понять особенности использования регулярных выражений в Java, научиться работать с строками и применять регулярные выражения для обработки строк в программах.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Регулярные выражения и их использование в Java программах для обработки строк

Регулярные выражения – эта система обработки текста, основанная на специальной системе записи образцов для поиска. Образец (pattern), задающий правило поиска, по-русски также иногда называют «шаблоном», «маской». Сейчас регулярные выражения используются многими текстовыми редакторами и утилитами для поиска и изменения текста на основе выбранных правил. Язык программирования Java также поддерживает регулярные выражения для работы со строками.

Основными классами для работы с регулярные выражения являются класс java.util.regex.Pattern и класс java.util.regex.Matcher.

Для определения шаблона применяются специальные синтаксические конструкции.

Класс Pattern

Класс java.util.regex.Pattern применяется для определения регулярных выражений, для которого ищется соответствие в строке, файле или другом объекте представляющем собой некоторую последовательность символов. Этот класс используется для простой обработки строк. Для более сложной обработки строк используется класс Matcher, рассматриваемый ниже.

В классе Pattern объявлены следующие методы:

- compile(String regex) – возвращает Pattern, который соответствует regex;
- matcher(CharSequence input) – возвращает Matcher, с помощью которого можно находить соответствия в строке input;
- matches(String regex, CharSequence input) – проверяет на соответствие строки input шаблону regex;
- pattern() – возвращает строку, соответствующую шаблону;
- split(CharSequence input) – разбивает строку input, учитывая, что разделителем является шаблон;
- split(CharSequence input, int limit) – разбивает строку input на не более чем limit частей.

С помощью метода matches() класса Pattern можно проверять на соответствие шаблону целой строки, но если необходимо найти соответствия внутри строки, например, определять участки, которые соответствуют шаблону, то класс Pattern не может быть использован. Для таких операций необходимо использовать класс Matcher.

Класс Matcher

С помощью класса java.util.regex.Matcher можно получить больше информации каждом соответствии.

Начальное состояние объекта типа Matcher не определено. Попытка вызвать какой-либо метод класса для извлечения информации о найденном соответствии приведет к возникновению ошибки IllegalStateException. Для того чтобы начать работу с объектом Matcher нужно вызвать один из его методов:

- matches() – проверяет, соответствует ли вся строка шаблону;
- lookingAt() – пытается найти последовательность символов, начинающуюся с начала строки и соответствующую шаблону;
- find() или find(int start) – пытается найти последовательность символов, соответствующих шаблону, в любом месте строки. Параметр start указывает на начальную позицию поиска.

Иногда необходимо сбросить состояние объекта класса Matcher в исходное, для этого применяется метод reset() или reset(CharSequence input), который также устанавливает новую последовательность символов для поиска.

Для замены всех подпоследовательностей символов, удовлетворяющих шаблону, на заданную строку можно применить метод replaceAll(String replacement).

Для того чтобы ограничить поиск границами входной последовательности применяется метод region(int start, int end), а для получения значения этих границ – regionEnd() и regionStart(). С регионами связано несколько методов:

useAnchoringBounds(boolean b) – если установлен в true, то начало и конец региона соответствуют символам ^ и $ соответственно;

hasAnchoringBounds() – проверяет закрепленность границ.

В регулярном выражении для более удобной обработки входной последовательности применяются группы, которые помогают выделить части найденной подпоследовательности. В шаблоне они обозначаются скобками «(» и «)». Номера групп начинаются с единицы. Нулевая группа совпадает со всей найденной подпоследовательностью. Далее приведены методы для извлечения информации о группах:

end() – возвращает индекс последнего символа подпоследовательности, удовлетворяющей шаблону;
end(int group) – возвращает индекс последнего символа указанной группы;
group() – возвращает всю подпоследовательность, удовлетворяющую шаблону;
group(int group) – возвращает конкретную группу;
groupCount() – возвращает количество групп;
start() – возвращает индекс первого символа подпоследовательности, удовлетворяющей шаблону;
start(int group) – возвращает индекс первого символа указанной группы;
hitEnd() – возвращает истину, если был достигнут конец входной последовательности.

Следующий пример показывает использование возможностей классов Pattern и Matcher, для поиска, разбора и разбиения строк.

```
import java.util.regex.*;
public class DemoRegular {
public static void main(String[] args) {
// проверка на соответствие строки шаблону
Pattern p1 = Pattern.compile("a*y");
Matcher m1 = p1.matcher( "aaay" );
boolean b = m1.matches();
System.out.println(b);
// поиск и выбор подстроки, заданной шаблоном
String regex = "(\\w+)@(\\w+\\.)(\\w+)(\\.\\w+)*" ;
String s = "адреса эл.почты: mymail@tut.by и rom@bsu.by";
Pattern p2 = Pattern. compile (regex);
Matcher m2 = p2.matcher(s);
while (m2.find()) {
System.out.println("e-mail: " + m2.group());
}
// разбиение строки на подстроки с применением шаблона в качестве
// разделителя
Pattern p3 = Pattern.compile("\\d+\\s?");
String[] words = p3.split("java5tiger 77 java6mustang");
for (String word : words)
System.out.println(word);
}
}
}
```

В результате работы программы будет выведено:
true
e - mail : mymail @ tut. by
e-mail: rom@bsu.by
java
tiger
java
mustang

Следующий пример демонстрирует возможности использования групп, а также собственных и неполных квантификаторов.

```
import java.util.regex.*;
public class Groups {
public static void main(String[] args) {
String input = "abdcxyz";
myMatches("([a-z]*)([a-z]+)", input);
myMatches("([a-z]?)([a-z]+)", input);
myMatches("([a-z]+)([a-z]*)", input);
myMatches("([a-z]?)([a-z]?)", input);
}
public static void myMatches(String regex,
String input) {
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(input);
if(matcher.matches()) {
System.out.println("First group: "
+ matcher.group(1));
System.out.println("Second group: "
+ matcher.group(2));
} else
System.out.println("nothing");
System.out.println();
}}
```

В результате работы программы будет выведено:

First group: abdcxy
Second group: z
First group: a
Second group: bdcxyz
First group: abdcxyz
Second group: nothing

В первом случае к первой группе (First group) относятся все возможные символы, но при этом остается минимальное количество символов для второй группы (Second group). 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() – первой 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.

Класс StringTokenizer

Этот класс предназначен для разбора строки по лексемам (tokens). Строка которую необходимо разобрать передается в качестве параметра конструктору StringTokenizer(String str). Определено еще два перегруженных конструктора, которым дополнительно можно передать строку-разделитель лексем StringTokenizer(String str,String delim) и признак возврата разделителя лексем StringTokenizer(String str,String delim,Boolean retirnDelims). Разделителем лексем по умолчанию служит пробел.

Пример:

```
public class Test {

  public Test() {
}
public static void main(String[] args) {
Test test = new Test();
String toParse = “word1;word2;word3;word4”;
StringTokenizer st = new StringTokenizer(toParse,";");
while(st.hasMoreTokens()){
System.out.println(st.nextToken());
}
}
}
```

Вывод:

```
word1
word2
word3
word4
```

Задачи

1. Необходимо реализовать консольное приложение, позволяющее манипулировать строкой, разбив ее на элементы путем использования регулярных выражений.
2. Написать регулярное выражение, определяющее является ли данная строка строкой "abcdefghijklmnopqrstuv18340" или нет.

a) пример правильных выражений: abcdefghijklmnopqrstuv18340
b) пример неправильных выражений: abcdefghijklmnoasdfasdpqrstuv18340.

1. Дан текст со списками цен. Извлечь из него цены в USD, RUB, EU.
   – пример правильных выражений: 25.98 USD.
   – пример неправильных выражений: 44 ERR, 0.004 EU.
2. Дан текст, необходимо проверить есть ли в тексте цифры, за которыми не стоит знак «+».
   – пример правильных выражений: (1 + 8) – 9 / 4.
   – пример неправильных выражений: 6 / 5 – 2 * 9 .
3. Написать регулярное выражение, определяющее является ли данная строчка датой в формате dd/mm/yyyy. Начиная с 1900 года до 9999 года.
   – пример правильных выражений: 29/02/2000, 30/04/2003, 01/01/2003.
   – пример неправильных выражений: 29/02/2001, 30-04-2003, 1/1/1899.
4. Написать регулярное выражение, определяющее является ли данная строчка допустимым (корректным) e-mail адресом согласно RFC под номером 2822.
   – пример правильных выражений: user@example.com, root@localhost
   – пример неправильных выражений: myhost@@@com.ru, @my.ru, Julia String.
5. Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов. Где символом может быть цифр, английская буква, и знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

– пример правильных выражений: F032_Password, TrySpy1.
– пример неправильных выражений: smart_pass, A007.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Общая технология работы с регулярными выражениями.
2. Классы для работы с датой и временем в Java 8 в формате 24
3. Общие сведения для работы с датой и временем в Java 8.

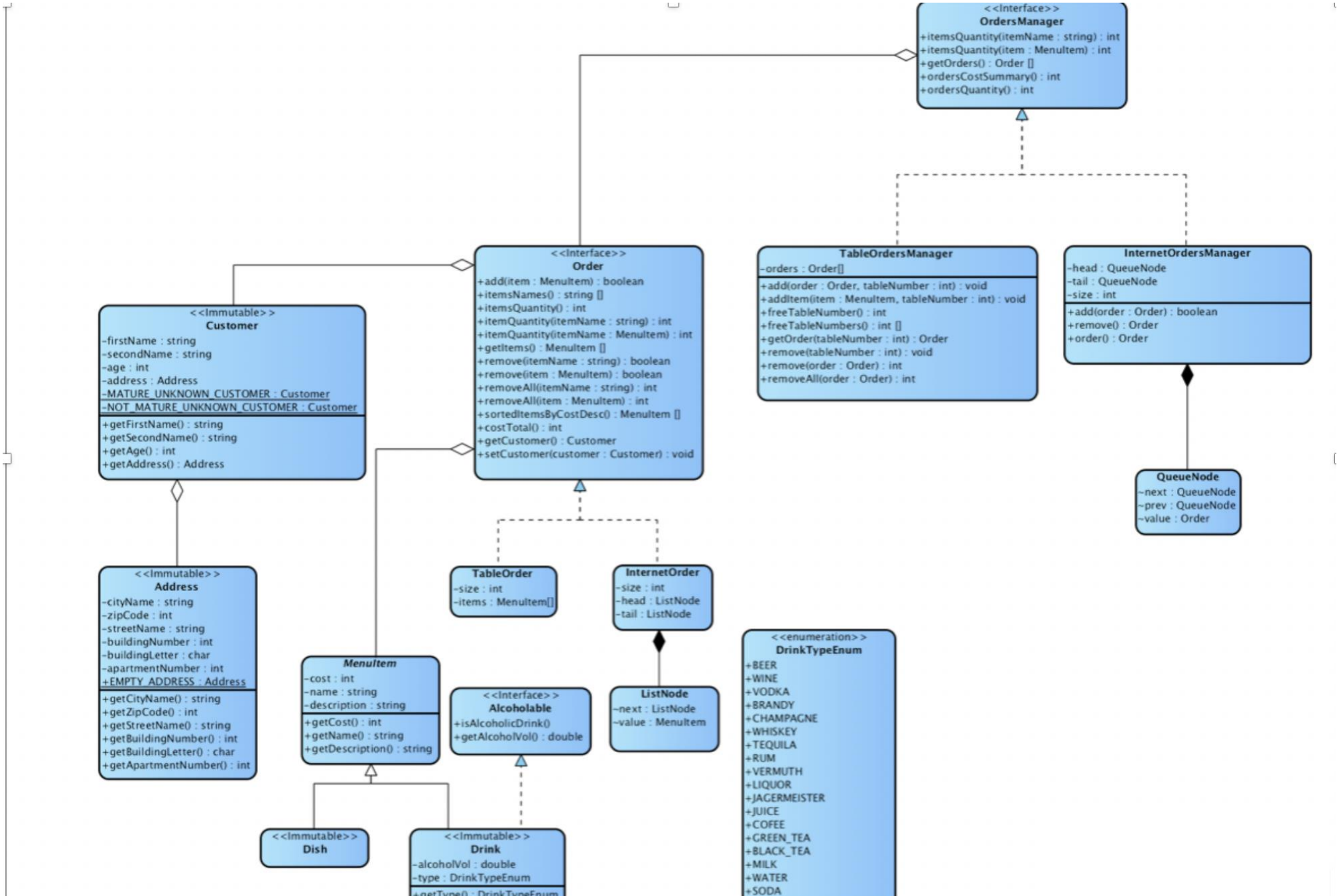
ССЫЛКИ ДЛЯ ЧТЕНИЯ

1. The Java Language Specification, Java SE 7 Edition [электронный документ] :
2. https://javarush.ru/groups/posts/regulyarnye-vyrazheniya-v-java
3. https://proselyte.net/tutorials/java-core/regular-expressions/http://docs.oracle.com/javase/specs
4. http://www.javenue.info/post/43
5. The Java Tutorials, http://docs.oracle.com/javase/tutorial/index.html
6. Bloch, Joshua. Effective JavaTM. Second Edition. – Addison-Wesley, 2008.
7. Хабибулин И.Ш. Java 7 // И.Ш. Хабибулин. – СПб.: БХВ-Петербург, 2012. – 768 с.: ил (В подлиннике).

Лабораторная работа № 16

Задание 1 Разработка кода по UML диаграмме с использованием классов, разработанных в практической работе №16

Задание 2. Разработка интерфейса пользователя для интерактивного взаимодействия.


<<Interface>>
OrdersManager
+itemsQuantity(itemName : string) : int
+itemsQuantity(item : MenuItem) : int
+getOrders() : Order []
+ordersCostSummary() : int
+ordersQuantity() : int
<<Interface>>
Order
+add(item : MenuItem) : boolean
+itemsNames() : string []
+itemsQuantity() : int
+itemQuantity(itemName : string) : int
+itemQuantity(itemName : MenuItem) : int
+getItems() : MenuItem []
+remove(itemName : string) : boolean
+remove(item : MenuItem) : boolean
+removeAll(itemName : string) : int
+removeAll(item : MenuItem) : int
+sortedItemsByCostDesc() : MenuItem []
+costTotal() : int
+getCustomer() : Customer
+setCustomer(customer : Customer) : void
TableOrdersManager
-orders : Order[]
+add(order : Order, tableNumber : int) : void
+addItem(item : MenuItem, tableNumber : int) : void
+freeTableNumber() : int
+freeTableNumbers() : int []
+getOrder(tableNumber : int) : Order
+remove(tableNumber : int) : void
+remove(order : Order) : int
+removeAll(order : Order) : int
InternetOrdersManager
-head : QueueNode
-tail : QueueNode
-size : int
+add(order : Order) : boolean
+remove() : Order
+order() : Order
QueueNode
~next : QueueNode
~prev : QueueNode
~value : Order
<<Immutable>>
Customer
-firstName : string
-secondName : string
-age : int
-address : Address
-MATURE_UNKNOWN_CUSTOMER : Customer
-NOT_MATURE_UNKNOWN_CUSTOMER : Customer
+getFirstName() : string
+getSecondName() : string
+getAge() : int
+getAddress() : Address
<<Immutable>>
Address
-cityName : string
-zipCode : int
-streetName : string
-buildingNumber : int
-buildingLetter : char
-apartmentNumber : int
+EMPTY_ADDRESS : Address
+getCityName() : string
+getZipCode() : int
+getStreetName() : string
+getBuildingNumber() : int
+getBuildingLetter() : char
+getApartmentNumber() : int
TableOrder
-size : int
-items : MenuItem[]
InternetOrder
-size : int
-head : ListNode
-tail : ListNode
ListNode
~next : ListNode
~value : MenuItem
MenuItem
-cost : int
-name : string
-description : string
+getCost() : int
+getName() : string
+getDescription() : string
<<Interface>>
Alcoholable
+isAlcoholicDrink()
+getAlcoholVol() : double
<<enumeration>>
DrinkTypeEnum
+BEER
+WINE
+VODKA
+BRANDY
+CHAMPAGNE
+WHISKEY
+TEQUILA
+RUM
+VERMUTH
+LIQUOR
+JAGERMEISTER
+JUICE
+COFEE
+GREEN_TEA
+BLACK_TEA
+MILK
+WATER
+SODA
<<Immutable>>
Dish
<<Immutable>>
Drink
-alcoholVol : double
-type : DrinkTypeEnum