Обработка строк

- Постоянство строк
- Объекты класса строк постоянны (не изменяемы) (immutable)-> каждый метод класса, изменяющий строку, на самом деле возвращает новую
- String oldString = "oldString"
- SomeChange("oldString"); //oldString по прежнему старая, если выведем, то будет что-то еще, но если обратимся к старой она будет старой
- String newString = SomeChange("oldString") // newString будет старой с изменениями
- Оператор + -> перегружен, можно использовать для объединения строк. Внутри выполняется с помощью StringBuilder, чтоб не создавать переменную строку для присвоения значений
- При использовании StingBuilder - внутри создание только одного объекта ++ заранее выделение объекта нужного размера

- непреднамеренная рекурсия
- Все объекты содержать метод toString, контейнеры тоже
- toString переопр, перебирает каждый объект + вызывает для
- каждого из них toString
- Если переопределить toString и поместить не строку туда,\
- то будет брать toString у объекта, если переопределить и поместить сам объект то
- буде рекурсия

Метод	Аргументы, перегрузка	Использование
Конструктор	Перегруженные версии: по умолчанию, String, StringBuilder, StringBuffer, массивы char, массивы byte	Создание объектов String
length()		Количество символов в String
charAt()	Индекс типа int	Символ (char) в заданной позиции строки
getChars(), getBytes()	Начало и конец копируемого участка, приемный массив, индекс в приемном массиве	Копирование блоков char и byte во внешний массив
toCharArray()		Создает массив char[], содержащий все символы String
equals(), equalsIgnoreCase()	Объект String для сравнения	Проверка равенства содержимого двух объектов String
compareTo()	Объект String для сравнения	Отрицательное число, нуль или положительное число в зависимости от лексикографического порядка String и аргумента. Символы верхнего и нижнего регистра не равны!
contains()	Искомая последовательность CharSequence	Результат равен true, если аргумент содержится в String
contentEquals()	CharSequence или StringBuffer для сравнения	Результат равен true при точном совпадении с аргументом
equalsIgnoreCase()	Объект String для сравнения	Результат равен true при равенстве содержимого без учета регистра символов
regionMatches()	Смещение в текущем объекте String, другой объект String, смещение и длина сравниваемого участка. Перегруженная версия добавляет признак игнорирования регистра	Результат типа boolean указывает, совпадают ли участки
startsWith()	Объект String, который может быть префиксом текущего объекта String	Результат типа boolean указывает, начинается ли объект String с аргумента
endsWith()	Объект String, который может быть суффиксом текущего объекта String	Результат типа boolean указывает, является ли аргумент суффиксом
indexOf(), lastIndexOf()	Перегруженные версии: char, char и начальный индекс, String, String и начальный индекс	Возвращает –1, если аргумент не найден в текущем объекте String; в противном случае возвращает индекс, по которому начинается аргумент. Метод lastIndexOf() осуществляет поиск в обратном направлении от конца строки
substring() (также subSequence())	Перегруженные версии: начальный индекс; начальный и конечный индексы	Возвращает новый объект String с заданным набором символов
concat()	Объект String для конкатенации	Возвращает новый объект String, состоящий из символов исходного объекта String, за которыми следуют символы аргумента
replace()	Искомый символ и новый символ, заменяющий его. Также возможна замена CharSequence на CharSequence	Возвращает новый объект String с внесенными изменениями. Если совпадения не найдены, используется старый объект String
toLowerCase(), toUpperCase()		Возвращает новый объект String, полученный изменением регистра символов. Если изменения отсутствуют, используется старый объект String
trim()		Возвращает новый объект String, в котором с обоих концов удалены пропуски. Если изменения отсутствуют, используется старый объект String
valueOf()	Перегруженные версии: Object, char[], char[] со смещением и количеством, boolean, char, int, long, float, double	Возвращает объект String с символьным представлением аргумента
intern()		Производит одну и только одну ссылку на String для каждой уникальной последовательности символов

- Форматирование вывода
- Форматирование спецификатора - подстановка значения вместо спецификатора
- Новая функциональность форматирования - класс Fromatter
- Форматные спецификаторы
- %[аргумент_индекс$][флаги][ширина][.точность]преобразование
- Ширина - управляет минимальным размером поля
- Formatter - гарантирует. Что поле занимает не менее указанного количества символов, тип флаги - опр выравнивание
- Поле точность - задает максимальное значение. Для строк точность - макс знач символов, для вещ - количество цифр, точность не распространяется на целые числа
- Символы преобразования

Чаще всего на практике используются следующие преобразования

Символы преобразования	
d	Целое число (десятичное)
c	Символ Юникода
b	Логическое значение
s	Строка
f	Вещественное число (в десятичной записи)
e	Вещественное число (в экспоненциальной записи)
x	Целое число (шестнадцатеричное)
h	Хеш-код (в шестнадцатеричной записи)
%	Литерал «%»

- * преобр б рабоатает со всеми, если не булеан значение, то
- тру если не null
- String.format()- работает как класс Formatted, но возвращает строку
- Нужен, исли нужно использовать форматирование один раз
- Вывод файла в шестнадцетиричном виде
- на вход массив байтов
- для каждого из них, если счетчик кратен 16, то добавляем счетчик в 16-ричной
- и слэш, добавляем значение байта в 16-ти ричной

- Регулярные выражения
- Позволяют создавать шоблоны для поиска текста, обрабатывать его в дальнейшем
- Чтобы указать, что перед числом может быть(а может не быть) знак минус - -?
- Целое число - последовательность из цифр =, цифра - /d, просто d - //d
- \\\\ - просто \, в обозначениях новой строки - \n\t
- "необязательный минус, потом одна\несколько цифр" - -?\\d+
- Вариант ипользования регулярных выражений - можно проверить содержит ли строка регул выражения\
- с помощью matches
- в регул выражениях или через |
- экранировка символа - \\ чтоб ничего не значил
- split - может разбивать строку по заданным регулярным выражениям
- можно разделить и преобразовать в массив слов
- W - не символ слова, маленькая - символ слова
- replace - замена, тоже можно использовать регулярные выражения для замены

- Полный список регулярных выражений в описании класса Pattern

Символы	
B	Символ B
\xhh	Символ с шестнадцатеричным кодом 0xhh
\uhhhh	Символ Юникода с шестнадцатеричным представлением 0xhhhh
\t	Табуляция
\n	Новая строка
\r	Возврат курсора
\f	Подача страницы
\e	Escape

Мощь регулярных выражений начинает проявляться при определении символьных классов. Несколько типичных способов создания символьных классов, а также некоторые заранее определенные классы:

Символьные классы	
.	Любой символ
[abc]	Любой из символов a, b и c (то же, что a|b|c)
[^abc]	Любой символ, кроме a, b и c (отрицание)
[a-zA-Z]	Любой символ от a до z и от A до Z (диапазон)
[abc[hij]]	Любой из символов a, b, c, h, i, j (то же, что a|b|c|h|i|j) (объединение)
[a-z&&[hij]]	Символ h, i или j (пересечение)
\s	Пропуск (пробел, табуляция, новая строка, подача страницы, возврат курсора)
\S	Символ, не являющийся пропуском ([^\s])
\d	Цифра [0-9]
\D	Не цифра [^0-9]
\w	Символ слова [a-zA-Z_0-9]
\W	Символ, не являющийся символом слова [^\w]

Логические операторы	
XY	X, за которым следует Y
X|Y	X или Y
(X)	*Сохраняющая группировка.* Позднее в выражении к i-й сохраненной группе можно обратиться при помощи записи \i

Привязка к границам	
^	Начало строки
$	Конец строки
\b	Граница слова
\B	Не граница слова
\G	Конец предыдущего совпадения

Квантификаторы

Описывает режим "поглощения" входного текста в шаблоны

Максимальные - используются по умолчанию. Подбирается максимально возможное количество возможных совпадений

Минимальный (с вопросительным знаком) ограничивается минимальным количеством символов, необходимых для соответствия шаблону

Захватывающие поддерживаются только в Java. Игнорирют множественные состояния, которые генерируются для возврата в случае неудачи. Если не находят ничего не возвращают поэтому эффективны

Максимальный	Минимальный	Захватывающий	Совпадает
X?	X??	X?+	X, один или ни одного
X*	X*?	X*+	X, нуль и более
X+	X+?	X++	X, один и более
X{n}	X{n}?	X{n}+	X, ровно n раз
X{n,}	X{n,}?	X{n,}+	X, не менее n раз
X{n,m}	X{n,m}?	X{n,m}+	X, не менее n, но не более m раз

- Pattern и Matcher
- 1 – создать регулярне выражение
- 2 - откомпилировать с помощью Pattern.compile(),
- 3 – matcher() с набором операций -> в аргументах строка, в которой надо искать
- Find – для поиска множественных совпадений шаблона в CharSequence
- LookingAt - успех, только если совпадение в начале входных данных, как и у matches, а у find не так

- Группы - части регулярного выражения в кругдых скобках, к которым можно обращаться по номеру 0 - совпадение всего выражения
- С Matcher -
- GroupCount - количество групп в шаблоне объекта Matcher

Group - группа 0, все совпадения от предыдущей операции

Group(int i) - группа с заданным номером от предыдущей операции, если нет, то нул

Start(int group) - начальный индекс группы, найденной в предыдущей операции поиска совпадений

Int end(int group) - индекс последнего символа группы

- Start и end - после успешного поиска метод старт - начальный индекс, метод end - индекс последнего символа совпадения
- При неуспешном поиске выбрасывает исключение
- Флаги шаблонов
- Паттерн может принимать флаги, управляющие процессом поиска совпадений

```
Pattern Pattern.compile(String regex, int flag)
```

Флаг compile()	Эффект
Pattern.CANON_EQ	Два символа считаются совпадающими в том (и только в том!) случае, если совпадают их полные канонические декомпозиции
Pattern.CASE_INSENSITIVE (?i)	По умолчанию режим поиска совпадения без учета регистра символов распространяется только на символы набора US-ASCII. Поиск без учета регистра символов с поддержкой Юникода включается указанием флага UNICODE_CASE вместе с этим флагом

Флаг compile()	Эффект
Pattern.COMMENTS (?x)	В этом режиме пропуски игнорируются, а встроенные комментарии, начинающиеся с #, игнорируются до конца строки
Pattern.DOTALL (?s)	В этом режиме метасимвол «точка» (.) совпадает с любым символом, включая завершитель строки. По умолчанию точка не совпадает с завершителями строк
Pattern.MULTILINE (?m)	В этом режиме выражения ^ и $ совпадают с началом и концом логических строк соответственно. ^ также совпадает с началом входной строки, а $ — с концом входной строки. По умолчанию эти выражения совпадают только в начале и в конце всей входной строки
Pattern.UNICODE_CASE (?u)	Поиск совпадения без учета регистра символов, включаемый флагом CASE_INSENSITIVE, осуществляется способом, совместимым со стандартом Юникод. По умолчанию поиск без учета регистра символов распространяется только на символы из набора US-ASCII
Pattern.UNIX_LINES (?d)	В этом режиме в поведении метасимволов ., ^ и $ распознается только завершитель строк \n

- Операции замены
- ReplaceFirst - заменяет первое совпадение
- ReplaceAll - заменяет все совпадения
- Matcher - может быть применен к новой символьной последовательности с помощью reset - перевод matcher в начало текущей последовательности

- Регулярные выражения\ввод-вывод
- Применять регулярные выражения можно для поиска совпадений в файле - на вход имя файла и выражение, задать matcher, для каждого слова перейти в начало (reset) пока находит добавлять группу и позицию

- Сканирование ввода - с помощью класса Scanner (раньше - как-то сложно через файлы и их чтение) В сканнер можно добавить файл
- Существует next для всех примитивов - можно разбивать строку с помощью этого
- Ограничители сканера
- По умолчанию разбивает по пропускам, но можно создать ограничитель ф форме регулярного выражения
- Scanner.useDelimiter("\\s,\\s*"); - в качестве разгран - запятая
- Можно вбить в класс сканер строку и просканировать с ограничителем

- Сканирование с использованием регулярных выражений
- Сканирование по сложным шаблонам для сложных данных

```
public class ThreatAnalyzer {
  static String threatData =
    "58.27.82.161@02/10/2005\n" +
    "204.45.234.40@02/11/2005\n" +
    "58.27.82.161@02/11/2005\n" +
    "58.27.82.161@02/12/2005\n" +
    "58.27.82.161@02/12/2005\n" +
    "[Next log section with different data format]";
  public static void main(String[] args) {
    Scanner scanner = new Scanner(threatData);
    String pattern = "(\\d+[.]\\d+[.]\\d+[.]\\d+)@" +
      "(\\d{2}/\\d{2}/\\d{4})";
    while(scanner.hasNext(pattern)) {
      scanner.next(pattern);
      MatchResult match = scanner.match();
      String ip = match.group(1);
      String date = match.group(2);
      System.out.format("Threat on %s from %s\n", date,ip);
    }
  }
} /* Output:
Threat on 02/10/2005 from 58.27.82.161
Threat on 02/11/2005 from 204.45.234.40
Threat on 02/11/2005 from 58.27.82.161
Threat on 02/12/2005 from 58.27.82.161
Threat on 02/12/2005 from 58.27.82.161
*///:~
```