



# **Аннотации**

## Аннотации

- ❑ **Аннотации** – это форма метаданных для предоставления данных о программе, которые не являются частью самой программы.
  - не имеют прямого влияния на работу кода, который они аннотируют.
  - были добавлены в Java 5.
- ❑ Аннотации имеют ряд применений:
  - информация для компилятора - аннотации могут быть использованы компилятором для обнаружения ошибок или подавления предупреждений;
  - обработка во время компиляции и во время развертывания - инструменты программного обеспечения могут обрабатывать информацию аннотации для генерации кода, XML-файлов и так далее;
  - обработка выполнения - некоторые аннотации доступны и во время выполнения.

## Аннотации

- ❑ Аннотации выглядят следующим образом:

**@Entity** (@Сущность)

- ❑ Символ-признак (@) и указывает компилятору, что далее следует аннотация.
- ❑ Аннотации могут быть применены к объявлениям:
  - классов,
  - полей,
  - методов,
  - других элементов программы.
- ❑ При использовании на объявлении, каждая аннотация в соответствии с соглашением отображается в отдельной собственной строке.

## Аннотации

- ❑ Предопределенные типы аннотаций, размещенные в **java.lang**:
  - **@Deprecated** – аннотируемый элемент устарел и должен быть заменен на новую форму;
  - **@Override** – аннотируемый метод должен переопределять метод суперкласса;
  - **@SuppressWarnings** – определенные предупреждения, выдаваемые компилятором, должны быть подавлены;
  - **@SafeVarargs** – применяется только к методам и конструкторам с переменным количеством параметров, которые объявлены как ***static*** или ***final***, и указывает, что никакие небезопасные действия, связанные с параметром переменного количества аргументов, недопустимы.

<https://docs.oracle.com/javase/tutorial/java/annotations/predefined.html>

## Аннотации

❑ Аннотация **@Override** сообщает компилятору, что элемент предназначен для переопределения элемента, объявленного в суперклассе:

- Нет требования использовать эту аннотацию, когда переопределяется метод, но она помогает избежать ошибок;
- Если метод, отмеченный **@Override**, не выполняет корректно переопределение метода одного из его суперкласса, то компилятор генерирует сообщение об ошибке.

1. **@Override**

2. **int** overriddenMethod() {

3. ...

4. }

## Аннотации

- ❑ Аннотация **@Deprecated** указывает, что отмеченный элемент является устаревшим и больше не используется;
- ❑ Компилятор генерирует предупреждение всякий раз, когда программа использует метод, класс или поле с аннотацией **@Deprecated**;
- ❑ Когда элемент является устаревшим, он также должен быть задокументированным с помощью тега **javadoc @deprecated**:  

```
/** * @deprecated * explanation of why it was  
    deprecated */  
@Deprecated static void deprecatedMethod() { //... }
```
- ❑ Использование признака (@) и комментарии **javadoc** и в аннотации не случайно: они связаны концептуально. Кроме того, тег **javadoc** начинается со строчной **d**, а аннотация начинается с прописной **D**.

## Аннотации

- ❑ Аннотация **@SuppressWarnings** сообщает компилятору о подавлении конкретного предупреждения, которое, в противном случае, будет генерироваться:

```
@SuppressWarnings("deprecation")
public static void main(String[] args) {
    long dt = Date.parse(args[0]);
}
```

- ❑ Каждое предупреждение компилятора относится к категории:
  - deprecation;
  - unchecked (может возникнуть при взаимодействии со старым кодом, написанным перед появлением обобщений).
- ❑ Для подавления нескольких категорий предупреждений, используйте следующий синтаксис:

```
@SuppressWarnings({"unchecked", "deprecation"})
```

## Аннотации

- ❑ Аннотация **@SafeVarargs**, применяемая только к методу или конструктору, утверждает, что код не выполняет потенциально опасные операции с его параметром переменной длины.
- ❑ Когда используется этот тип аннотации, предупреждения категории *unchecked*, касающиеся использования переменной длины, подавляются.
- ❑ Если эта аннотация будет сопровождать метод или конструктор, которые не содержат параметр переменной длины или содержат параметр переменной длины, но он не *static* или *final*, то будет ошибка компиляции.

[https://blogs.oracle.com/darcy/entry/project\\_coin\\_safe\\_varargs](https://blogs.oracle.com/darcy/entry/project_coin_safe_varargs)



# ПОЛЬЗОВАТЕЛЬСКИЕ АННОТАЦИИ

- ❑ Главной задачей аннотаций является статическое расширение классов (*именно классов, а не объектов*), путём добавления метаданных в класс, без изменения его методов и свойств.
- ❑ Многие современные технологии требуют большого количества вспомогательных файлов, конфигурационных файлов и т.п. (особенно Java EE).
  - чтобы не создавать всё это руками - можно применить аннотацию (определить, что этот класс является веб-сервисом и «натравить» на класс процессор аннотаций - он превратит эту аннотацию в набор классов и конфигурационных файлов).

<http://docs.oracle.com/javase/1.5.0/docs/guide/language/annotations.html>

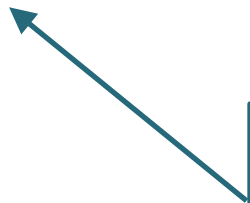
## Аннотации

- ❑ Аннотация **@interface** применяется для описания пользовательской аннотации:
  - Объявления типа аннотации похоже на обычное объявление интерфейса – состоят только из объявления методов;
  - Метод ведет себя подобно полю – это параметр аннотации;
  - Тип возвращаемого значения ограничивается примитивами, **String**, **Class**, **enum**, аннотациями и массивами указанных типов;
  - Метод не может иметь каких-либо параметров или компонента **throws**;
  - Методы могут иметь значения по умолчанию;
  - Автоматически аннотации расширяют интерфейс **Annotation** и не могут иметь компонент **extends**.
  - Объявление типа аннотации само тоже аннотировано
    - ✓ такие аннотации называются мета-аннотациями.

## Аннотации

### Пример 1,

```
@Target(value = ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MyAnnotation {
    String param1() default "some def value";
    String param2();
}
```



Определение  
значения по  
умолчанию

## Аннотации

- ❑ Аннотация **@Target** ограничивает в том, к каким элементам исходного кода могут быть применены пользовательские аннотации:
- ❑ Предназначена для применения только в качестве аннотации к другим аннотациям.
  - ElementType.ANNOTATION\_TYPE
  - ElementType.CONSTRUCTOR
  - ElementType.FIELD
  - ElementType.LOCAL\_VARIABLE
  - ElementType.METHOD
  - ElementType.PACKAGE
  - ElementType.PARAMETER
  - ElementType.TYPE

## Аннотации

- ❑ Аннотация **@Retention** определяет, как маркируемые ее аннотации хранятся:
  - **RetentionPolicy.SOURCE** - аннотация сохраняется только на уровне исходного кода и игнорируется компилятором;
  - **RetentionPolicy.CLASS** - аннотация сохраняется компилятором во время компиляции, но игнорируется виртуальной машиной Java (JVM);
  - **RetentionPolicy.RUNTIME** - аннотация удерживается JVM так, что она может быть использована в среде исполнения.
- ❑ Предназначена для применения только в качестве аннотации к другим аннотациям;
- ❑ Если политика удержания не указывается в объявлении аннотации, по умолчанию используется **CLASS**;
- ❑ Аннотация для маркировки локальных переменных в файл **.class** не сохраняется.

## Аннотации

- ❑ Аннотация **@Inherited** указывает, что тип аннотации может быть унаследован от суперкласса:
  - по умолчанию, если она не указывается, то наследование отсутствует;
- ❑ Когда пользователь запрашивает тип аннотации, а класс не имеет аннотации, то запрашивается "суперкласс" класс для получения типа аннотации;
- ❑ Эта аннотация применяется только к объявлению типа аннотации.

## Аннотации

- ❑ Аннотация **@Documented** показывает, что всякий раз, когда используется указанная аннотация, ее элементы должны быть задокументированы с помощью инструмента **javadoc**.
- ❑ По умолчанию, аннотации не включаются в **Javadoc**.

## Аннотации

- ❑ В основном аннотации спроектированы для использования инструментами разработки и развертывания, однако их можно опрашивать и во время выполнения с помощью рефлексии.
- ❑ Через объекты типа **Class**, **Method**, **Field** и **Constructor** можно получить аннотацию, связанную с соответствующим объектом, с помощью метода *getAnnotation(Class<A> an)*;
  - Если аннотация не найдена, то возвращается **null**;
- ❑ Методы *getAnnotation()*, *getAnnotations()*, *getDeclaredAnnotations()*, *isAnnotationPresent()* – определены интерфейсом **java.lang.reflect.AnnotatedElement**;



## Аннотации

- ❑ **Аннотация-маркер** – это специальный вид аннотаций, который не содержит членов:
  - назначение такой аннотации – отметить объявление;
  - например: `@Override`, `@SafeVarargs`, `@Inherited`.
  
- ❑ **Одночленная аннотация** – это аннотация, которая содержит только один член и допускает сокращенную форму применения:
  - член аннотации должен иметь имя **value**;
  - однако, одночленную аннотацию можно применять и с использованием аннотации с другими членами, однако эти члены должны иметь значения по умолчанию;
  - например: `@Target`, `@Retention`.

## Аннотации

### Пример 2:

**@Retention**(RetentionPolicy.***RUNTIME***)

**@interface** **MySingle** {

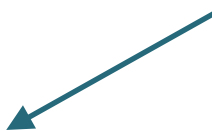
int value();

}

**class** Single {

**@MySingle**(100)

**public static void** method() { ..... }



Определение  
значения параметра  
без указания имени

## Аннотации

### Пример 3:

```
public enum Color { BLACK, WHITE, RED, GREEN, BLIE }
```

```
@Target(ElementType.METHOD)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface MyInfo {  
    Color[] value();  
    String info() default "Color -> GREEN";  
}
```

```
class Sample {  
    @MyInfo({Color.BLACK, Color.WHITE})  
    public static void method() { //..... }  
}
```

Установка нескольких значений, второй параметр – значение по умолчанию

## Аннотации

### Пример 4:

```
public enum Color { BLACK, WHITE, RED, GREEN, BLIE }
```

```
@Target(ElementType.METHOD)
```

```
@Retention(RetentionPolicy.RUNTIME)
```

```
public @interface MyInfo {  
    Color[] value();  
    String info();  
}
```

Второй параметр не  
имеет значения по  
умолчанию

```
class Sample {  
    @MyInfo(value = Color.BLACK, info = "Color default -> RED")  
    public void method1() { //..... }  
}
```

Требуется явно  
указывать имена и  
значения

# **Регулярные выражения**

## Регулярные выражения

- ❑ **Регулярное выражение** (regular expression) – это текстовая строка, которая описывает шаблон некоторых данных (инструмент, используемый для обработки текста).

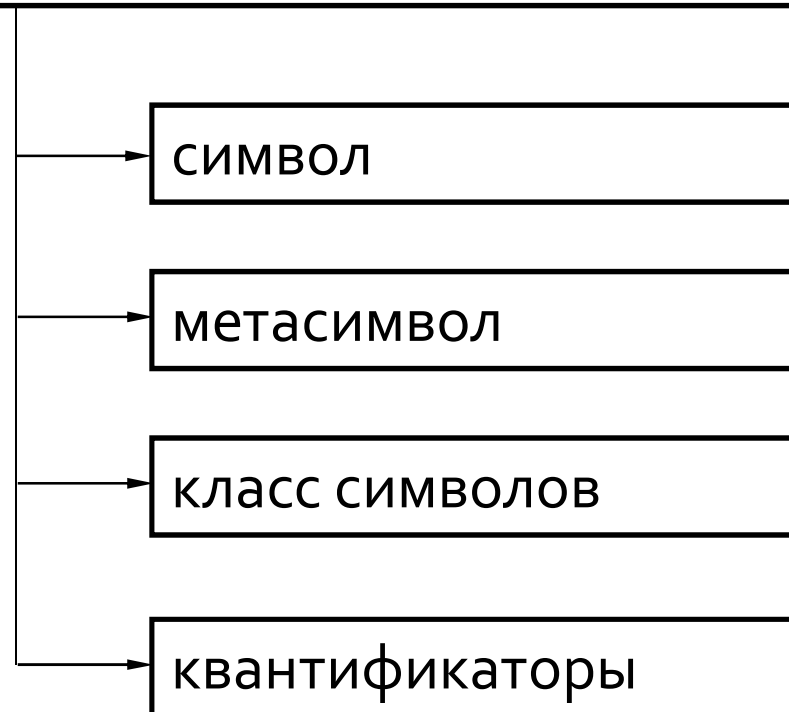
**RegExp**  
**^\+[a-z]{1,8}\$**

- ❑ Назначение:

- поиск информации (слово; слова, начинающиеся с некоторых символов; слова, заканчивающиеся некоторыми символами и т.д.);
- синтаксический анализ текста.

## Регулярные выражения

### Компоненты регулярного выражения



## Регулярные выражения

<символ> ::= любой символ за исключением специальных  
(‘а’, ‘№’, ‘Р’, ‘~’, ‘5’, ‘\45’, ‘\023’, ‘\ua376)

<метасимвол> ::= ^ или \A                      - начало строки  
\$ или \Z или \z                      - конец строки  
< и > или \b                      - ограничивают слово  
.  
|                      - выбор  
( )                      - ограничение при выборе,  
                            группировка для повторений,  
                            запоминание для повторного поиска  
\a, \n, \r, \t, \v, \e                      - управляющие символы  
\1, \2, ...                      - обратные ссылки



Метасимвол – это директивы



## Регулярные выражения

$\langle \text{класс символов} \rangle ::= [\text{любой из перечисленных символов}] |$   
 $[\text{^любой отличный от перечисленных}] |$   
 $[\text{символ} - \text{символ}]$



Класс символов – это набор символов, из которых может соответствовать любой.

*Например,*

[dgvhynef]

[4-7] или [4567]

[^a-z]



Тоже самое: или 4 или 5 или 6 или 7

## Регулярные выражения

### Метасимволы, которые заменяют классы символов

**\d** - цифры [0-9];

**\D** - не цифры [^0-9];

**\s** - символ пробела [\t\n\x0b\f\r];

**\S** - не пробельные символы [^\s];

**\w** - буквенные и цифровые символы [a-zA-Z\_0-9];

**\W** - не буквенные и цифровые символы [^\w].

## Регулярные выражения

<квантификаторы> ::=

- |            |   |
|------------|---|
| ?          | - ноль или один раз                                 |
| +          | - один или более                                    |
| *          | - ноль или более                                    |
| {min, max} | - диапазон повторений от <i>min</i> и по <i>max</i> |
| *?         | - найти наименьшее совпадение                       |

❑ Диапазон может указываться вариативно:

- ✓ {min,} - не меньше min
- ✓ {, max} - не больше max
- ✓ {number} - точно number



Квантификатор— это оператор повторения

## Регулярные выражения

- ❑ Если необходимо использовать обозначение метасимвола или квантификатора в качестве обычного символа, тогда применяется экранирование:

1. \<метасимвол> (например: \\*, \+, \., \?)

2. [<метасимвол>] (например: [+], [?], [\*])



Метасимвол **\b** интерпретируется по разному, в зависимости от контекста использования:

- ✓ если указывается в классе символов, то обозначает символ «backspace» ( [\b] );
- ✓ если указывается вне класса символов, то означает границу слова (*т.е. ограничивает слова, состоящие из символов [a-zA-Z0-9\_]*).

## Регулярные выражения

### *Правило формирования регулярного выражения:*

$\langle \text{выражение} \rangle ::= \langle \text{компонент} \rangle \mid \langle \text{компонент} \rangle \langle \text{выражение} \rangle$

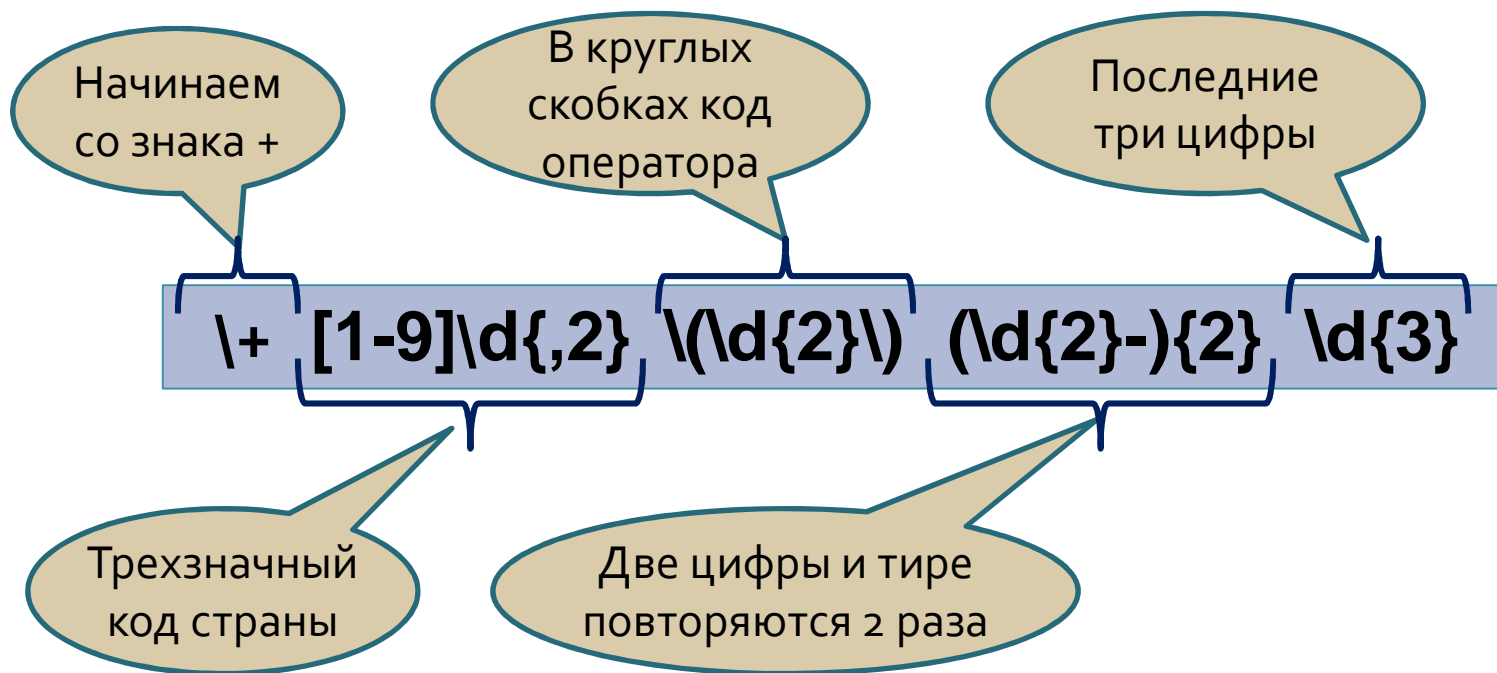
$\langle \text{компонент} \rangle ::= \langle \text{коэффициент} \rangle \mid \langle \text{коэффициент} \rangle \langle \text{компонент} \rangle$

$\langle \text{коэффициент} \rangle ::= \langle \text{элемент} \rangle \mid \langle \text{элемент} \rangle \langle \text{квантификатор} \rangle$

$\langle \text{элемент} \rangle ::= \langle \text{символ} \rangle \mid \langle \text{метасимвол} \rangle \mid (\langle \text{выражение} \rangle) \mid \langle \text{класс символов} \rangle$

## Регулярные выражения

Например, форма указания мобильного номера телефона:



**+380(99)22-44-888**  
**+380(67)98-54-321**

## Регулярные выражения

*Например,*

`[a-zA-Z_][a-zA-Z0-9_]*` - описание идентификатора

`((\+|-)?[1-9][0-9]*)|0` - описание целого значения

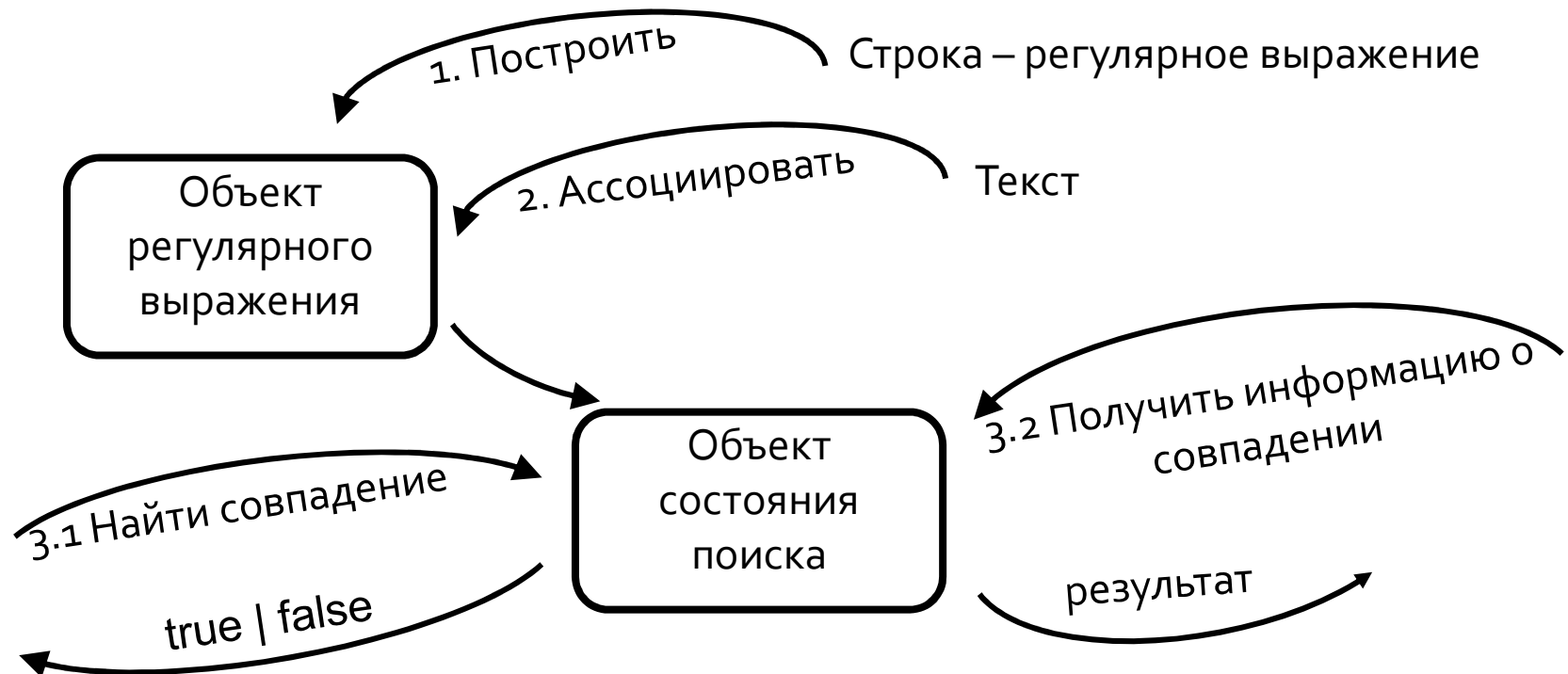
`(\+|-)?((([1-9][0-9]*)|0)?\.[0-9]+` - описание  
вещественного  
значения

*Задание:* приведите регулярное выражение для описания электронного адреса.

## Регулярные выражения

### Средства языка Java для работы с регулярными выражениями

- Для построения и работы используется пакет **java.util.regex**.





## Регулярные выражения

- ❑ Последовательность действий при работе с регулярными выражениями:
  1. Создать строку с регулярным выражением и откомпилировать ее во внутреннее представление (статический метод **compile()** класса **Pattern**).
  2. Ассоциировать регулярное выражение с текстом (метод **matcher()** класса **Pattern**).
  3. Проверка успешности совпадения (метод **find()** класса **Matcher**).
  4. Запрос данных (методы класса **Matcher**).
  5. Получение дополнительной информации о совпадении (методы класса **Matcher**).

## Регулярные выражения

- ❑ Метод **compile()** класса **Pattern** является перегруженным, кроме строки-шаблона можно указывать дополнительный набор флагов управления сопоставлением:

Флаг	Назначение
CASE_INSENSITIVE	Поиск соответствия без учета регистра для ASCII символов, т.е. строки: “abc”, “Abc” и “ABC” будут считаться соответствующими регулярному выражению “abc”
UNICODE_CASE	Поиск соответствия для символов, не входящих в ASCII
UNIX_LINES	Символ “\n” считается символом окончания строки, в которой выполняется поиск соответствия
MULTILINE	Если внутри текста, в которой выполняется поиск соответствия, есть символы “\n”, то считается то текст состоит из нескольких строк
LITERAL	Все символы шаблона, включая метасимволы, рассматриваются как обычные символы

## Регулярные выражения

Флаг	Назначение
DOTALL	Если в шаблоне есть метасимвол “.”, то ему будет соответствовать любой символ, включая символ “\n” (если отсутствует, то метасимвол “.” будет соответствовать любому символу, исключая символ “\n”)
COMMENTS	В строке шаблона, допустимы пробелы и комментарии, начинающиеся с символа “#” и до конца строки (при компиляции шаблона пробелы и комментарии будут проигнорированы)
CANON_EQ	Каноническая эквивалентность символов UNICODE (т.е. разные кодировки одного символа считаются идентичными).

## Регулярные выражения

Пример 10: поиск подстроки в зависимости от позиции

```
String str = "catddd cat cbdgw sewcat",  
text;
```

```
Pattern p0 = Pattern.compile("cat"),
```

Есть ли слово в тексте

```
p1 = Pattern.compile("^cat"),
```

Начинается ли текст с этого слова

```
p2 = Pattern.compile("cat$"),
```

Заканчивается ли текст этим словом

```
p3 = Pattern.compile("\\Acat\\Z");
```

Весь текст это слово

```
Matcher m0 = p0.matcher(str),
```

```
m1 = p1.matcher(str),
```

```
m2 = p2.matcher(str),
```

```
m3 = p3.matcher(str);
```

Ассоциировать регулярные  
выражения с текстом

## Регулярные выражения

Найти  
совпадение

```
System.out.println("Text -> " + str);
System.out.println("Find substring 'cat:");
while (m0.find()) {    text = m0.group();
    System.out.println(text + " from " + m0.start() + " to " + m0.end());
}
System.out.println("\nFind begin 'cat:");
while (m1.find()) {    text = m1.group();
    System.out.println(text + " from " + m1.start() + " to " + m1.end());
}
System.out.println("\nFind end 'cat:");
while (m2.find()) {    text = m2.group();
    System.out.println(text + " from " + m2.start() + " to " + m2.end());
}
System.out.println("\nFind 'cat:");
while (m3.find()) {    text = m3.group();
    System.out.println(text + " from " + m3.start() + " to " + m3.end());
}
```

Получить совпавшие данные

Вывести местоположение данных

## Регулярные выражения

*Результат:*

Text -> catddd cat cbdgw sewcat

Find substring 'cat':

cat from 0 to 3

cat from 7 to 10

cat from 20 to 23

Find begin 'cat':

cat from 0 to 3

Find end 'cat':

cat from 20 to 23

Find 'cat':

## Регулярные выражения

Пример 11: экранирование

String str;

String r1 = **"^\_(\\+|-)[A-K]\*\$"**,

r2 = **"^#[34567]+([\*]!)[D-K]\*# \$"**,

r3 = **"^[a-zA-Z]{1,5}\$"**;

Pattern p1 = Pattern.compile(r1),

p2 = Pattern.compile(r2),

p3 = Pattern.compile(r3);

Matcher m1, m2, m3;

Экранирование квантификатора +

Экранирование квантификатора \*

Сформировать объекты  
регулярных выражений

## Регулярные выражения

```
while (!(str = sc.next()).equals("n")) {  
    System.out.println("Text -> " + str);  
    m1 = p1.matcher(str);  
    if (m1.find())  
        System.out.println("True! Regular expression 1");  
    else {  
        m2 = p2.matcher(str);  
        if (m2.find())  
            System.out.println("True! Regular expression 2");  
        else {  
            m3 = p3.matcher(str);  
            if (m3.find())  
                System.out.println("True! Regular expression 3");  
            else  
                System.out.println("False!");  
        }  
    }  
}  
System.out.println("\nEnter string - > (for exit press key 'n') ");  
}
```

Проверка завершения ввода слов

Проверить 1-е регул.выражение на совпадение

Проверить 2-е регул.выражение на совпадение

Проверить 3-е регул.выражение на совпадение

Приглашения для ввода



## Регулярные выражения

Regular expression:

1) `^_(\+|-)[A-K]*$`

2) `^#[34567]+([*]!)[D-K]*#$`

3) `^[a-zA-Z]{1,5}$`

Enter string - > (for exit press key 'n')

`#555*EFF#`

True! Regular expression 2

Enter string - > (for exit press key 'n')

`get`

True! Regular expression 3

Enter string - > (for exit press key 'n')

`_+BCD`

True! Regular expression 1

Enter string - > (for exit press key 'n')

`_-SA`

False!

## Регулярные выражения

Пример 12: поиск любых повторений слов

```
String str = "the The nvfrthe the The ssdeer. Ssdeer, aaa; My " +  
            "namename is Peter. I'm travel by Europe, europe",
```

```
reg = "([a-z]+) [.,;!?]* \\1";
```

Два подряд стоящих одинаковых слов

```
Pattern p = Pattern.compile(reg,  
    Pattern.CASE_INSENSITIVE|Pattern.UNICODE_CASE);
```

Без учета регистра для ASCII символов

```
Matcher m = p.matcher(str);
```

Без учета регистра для не ASCII символов

```
System.out.println("Text -> "+str+"\nRegular expression ->"+reg+"\n");
```

```
while (m.find()) {
```

```
    String text = m.group();
```

```
    System.out.println(text + " from " + m.start() + " to " + m.end());
```

```
}
```



Два подряд стоящих `\` означают экранирование управляющего символа языка Java

## Регулярные выражения

Результат:

Text -> the The nvfr the The ssdeer. Ssdeer, aaa; My namename is  
Peter. I'm travel by Europe, europe

Regular expression -> ([a-z]+)[.,;!~?]\*\1

the The -> 0 to 7

the the -> 12 to 19

ssdeer. Ssdeer -> 24 to 38

aa -> 40 to 42

namename -> 48 to 56

Europe, europe -> 81 to 95



Если подряд стоящих одинаковых слов будет больше чем два, то созданное регулярное выражение это не отследит.

Например,

## Регулярные выражения

### Пример 13: поиск с заменой

```
String str = "Lena, Ludmila, Sveta, Lulu, Natalia, Liliana, Lu";
```

```
Pattern p = Pattern.compile("Lu.*?\\b");
```

```
Matcher m = p.matcher(str);
```

```
System.out.println("Исходная строка -> " + str);
```

```
str = m.replaceAll("Masha");
```

```
System.out.println("Измененная строка -> " + str);
```

Найти все слова с букв Lu

### *Результат:*

Исходная строка -> Lena, Ludmila, Sveta, Lulu, Natalia, Liliana, Lu

Измененная строка -> Lena, Masha, Sveta, Masha, Natalia, Liliana,  
Masha

## Регулярные выражения

Пример 14: поиск с заменой

```
String str = "This is my second java 45 project.\nIt is wonderful to  
learn polysemantics and arrays.\nThe weather is cold,  
like in winter, but we all are expecting for spring";
```

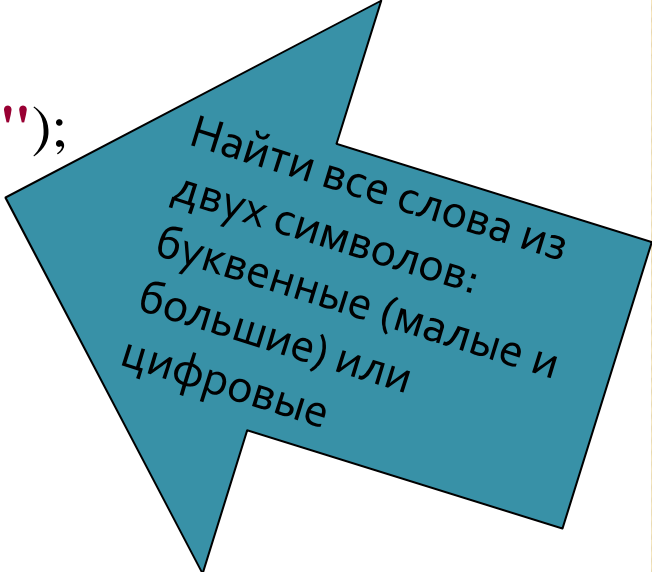
```
System.out.println("Before:\n" + str);
```

```
Pattern p = Pattern.compile("\\b[\\w]{2}\\b");
```

```
Matcher m = p.matcher(str);
```

```
str = m.replaceAll("lab2");
```

```
System.out.println("\nAfter:\n" + str);
```



Найти все слова из  
двух символов:  
буквенные (малые и  
большие) или  
цифровые

## Регулярные выражения

*Результат:*

### **Before:**

This **is my** second java **45** project.

**It is** wonderful **to** learn polysemantics and arrays.

The weather **is** cold, like **in** winter, but **we** all are expecting  
for spring

### **After:**

This **lab2 lab2** second java **lab2** project.

**lab2 lab2** wonderful **lab2** learn polysemantics and arrays.

The weather **lab2** cold, like **lab2** winter, but **lab2** all are  
expecting for spring

## Регулярные выражения

Пример 15: разбиение строки на лексемы

```
String str1 = "Один два,три!четыре;пять шесть.семь";  
Pattern p1 = Pattern.compile("[ ,!;.]");  
String s[] = p1.split(str1);  
System.out.println("Исходная строка -> " + str1);  
for (String i : s)  
    System.out.println("Лексема: " + i);
```



Можно использовать метод *split()* класса String, которому в параметрах передать разделители в виде регулярного выражения



## Регулярные выражения

*Результат:*

Исходная строка -> Один два,три!четыре;пять шесть.семь

Лексема: Один

Лексема: два

Лексема: три

Лексема: четыре

Лексема: пять

Лексема: шесть

Лексема: семь



## Регулярные выражения

- ❑ Если необходимо одноразовое сопоставление с шаблоном, тогда можно применить:
  1. Метод **matches()** класса **Pattern** (статический)
  2. Метод **matches()** класса **String**