

**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ**

Ордена Трудового Красного Знамени

**Федеральное государственное бюджетное образовательное учреждение
высшего образования**

«Московский технический университет связи и информатики»

Кафедра «Информационные технологии»

Отчет по лабораторной работе №5

по дисциплине «Информационные технологии и программирование» на
тему:

Регулярные выражения

Выполнил:

студент группы БВТ2302

Боданюк Алексей Павлович

Проверил:

Харрасов К. Р.

Москва 2024

Цель работы: изучить теорию, касающуюся регулярных выражений в Java.
Выполнить задания.

Задание:

Задание 1: Поиск всех чисел в тексте. Необходимо написать программу, которая будет искать все числа в заданном тексте и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска чисел и обрабатывать возможные ошибки.

Задание 2: Проверка корректности ввода пароля. Необходимо написать программу, которая будет проверять корректность ввода пароля. Пароль должен состоять из латинских букв и цифр, быть длиной от 8 до 16 символов и содержать хотя бы одну заглавную букву и одну цифру. При этом программа должна использовать регулярные выражения для проверки пароля и обрабатывать возможные ошибки.

Задание 3: Поиск заглавной буквы после строчной. Необходимо написать программу, которая будет находить все случаи в тексте, когда сразу после строчной буквы идет заглавная, без какого-либо символа между ними, и выделять их знаками «!» с двух сторон.

Задание 4: Проверка корректности ввода IP-адреса. Необходимо написать программу, которая будет проверять корректность ввода IP-адреса. IP-адрес должен состоять из 4 чисел, разделенных точками, и каждое число должно быть в диапазоне от 0 до 255. При этом программа должна использовать регулярные выражения для проверки IP-адреса и обрабатывать возможные ошибки.

Задание 5: Поиск всех слов, начинающихся с заданной буквы. Необходимо написать программу, которая будет искать все слова в заданном тексте, начинающиеся с заданной буквы, и выводить их на экран. При этом программа должна использовать регулярные выражения для поиска слов и обрабатывать возможные ошибки.

Ход работы:

У нас есть пример реализации метода в методических указаниях, воспользуемся тем кодом, немного усовершенствовав его.

```

Pattern pattern = Pattern.compile("\\d+(\\.\\d+)?");
Matcher matcher = pattern.matcher(text);

System.out.println("Found numbers:");
boolean found = false;
while (matcher.find()) {
    System.out.println(matcher.group());
    found = true;
}

```

Добавим в первую строку обозначение десятичной части числа как группу символов «()» и сделаем её необязательной с помощью «?». Так мы избавляемся от того, что не будут выводиться целые числа. Затем через try-catch выполним отлов ошибок, связанных с синтаксисом регулярного выражения, и проверку на отсутствие чисел как таковых:

```

try {
    Pattern pattern = Pattern.compile("\\d+(\\.\\d+)?");
    Matcher matcher = pattern.matcher(text);

    System.out.println("Found numbers:");
    boolean found = false;
    while (matcher.find()) {
        System.out.println(matcher.group());
        found = true;
    }
    if (!found) {
        System.out.println("В тексте не найдено чисел.");
    }
} catch (PatternSyntaxException e) {
    System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
}

```

Добавим проверку на пустую строку и проверим работоспособность кода:

```
NumberFinder.java x PasswordValidator.java CapitalAfterLowercase.java IPAddressValidator.java WordFind
1 import java.util.regex.*;
2
3 public class NumberFinder { no usages new *
4     public static void main(String[] args) { new *
5         String text = "The price of the product is $19.99, and it was $20 last week.";
6
7         if (text == null || text.isEmpty()) {
8             System.out.println("Текст для анализа отсутствует.");
9             return;
10        }
11
12        try {
13            Pattern pattern = Pattern.compile("\\d+(\\.\\d+)?");
14            Matcher matcher = pattern.matcher(text);
15
16            System.out.println("Found numbers:");
17            boolean found = false;
18            while (matcher.find()) {
19                System.out.println(matcher.group());
20                found = true;
21            }
22            if (!found) {
23                System.out.println("В тексте не найдено чисел.");
24            }
25        } catch (PatternSyntaxException e) {
26            System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
27        }
28    }
29 }
30
Terminal Local x
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java numberfinder.java
Found numbers:
19.99
20
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> |
```

Приступим к выполнению второго задания:

Аналогичным образом выполним реализацию через регулярное выражение

```
try {
    Pattern pattern = Pattern.compile("(?=.*[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}$");
    Matcher matcher = pattern.matcher(password);

    if (matcher.matches()) {
        System.out.println("Пароль корректен!");
    } else {
        System.out.println("Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру.");
    }
} catch (PatternSyntaxException e) {
    System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
}
```

Пояснение регулярного выражения

Регулярное выражение `^(?=.*[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}$` состоит из нескольких частей, каждая из которых добавляет определенные условия к паролю:

1. `^` — начало строки. Указывает, что регулярное выражение должно проверяться с начала строки.
2. `(?=.*[A-Z])` — позитивная опережающая проверка (*positive lookahead*). Она проверяет, что в пароле присутствует **хотя бы одна заглавная буква**. Детали:

- `(?=...)` — конструкция lookahead, которая проверяет, что после текущей позиции будет содержаться шаблон `...`, но не включает его в результат.
 - `.*` — любой символ в любом количестве.
 - `[A-Z]` — любая заглавная буква латинского алфавита.
3. `(?=.**\d)` — позитивная опережающая проверка, проверяющая, что в пароле есть хотя бы одна цифра.
 - `\d` — обозначение для любой цифры (0–9).
 4. `[A-Za-z\d]{8,16}` — основное тело шаблона, которое определяет длину и допустимые символы пароля.
 - `[A-Za-z\d]` — указывает, что пароль может содержать буквы латинского алфавита и цифры.
 - `{8,16}` — указывает, что длина пароля должна быть от 8 до 16 символов.
 5. `$` — конец строки. Указывает, что строка должна заканчиваться после соблюдения всех предыдущих условий, без лишних символов.

Также добавим проверку на пустой пароль и проверим работу кода:

The screenshot shows an IDE with several tabs: NumberFinder.java, PasswordValidator.java (active), CapitalAfterLowercase.java, IPAddressValidator.java, and WordFinder.java. The PasswordValidator.java file contains the following code:

```

1  import java.util.regex.*;
2
3  public class PasswordValidator { no usages new *
4  public static void main(String[] args) { new *
5      String password = "Password123";
6
7      if (password == null || password.isEmpty()) {
8          System.out.println("Пароль не может быть пустым.");
9          return;
10     }
11
12     try {
13         Pattern pattern = Pattern.compile("(?=.*[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}$");
14         Matcher matcher = pattern.matcher(password);
15
16         if (matcher.matches()) {
17             System.out.println("Пароль корректен!");
18         } else {
19             System.out.println("Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру.");
20         }
21     } catch (PatternSyntaxException e) {
22         System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
23     }
24 }
25
26

```

Below the code editor is a terminal window with the following output:

```

Terminal Local x + v
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java numberfinder.java
Found numbers:
19.99
20
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль корректен!
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5>

```

NumberFinder.javaPasswordValidator.javaCapitalAfterLowercase.javaIPAddressValidator.javaWordFinder.java

1import java.util.regex.*;

2

3public class PasswordValidator { no usages new *

4public static void main(String[] args) { new *

5String password = "";

6

7if (password == null || password.isEmpty()) {

8System.out.println("Пароль не может быть пустым.");

9return;

10}

11

12try {

13Pattern pattern = Pattern.compile("(?=.*[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}\$");

14Matcher matcher = pattern.matcher(password);

15

16if (matcher.matches()) {

17System.out.println("Пароль корректен!");

18} else {

19System.out.println("Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру");

20}

21} catch (PatternSyntaxException e) {

22System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());

23}

24}

25}

26

TerminalLocal × + ▾

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java numberfinder.java

Found numbers:

19.99

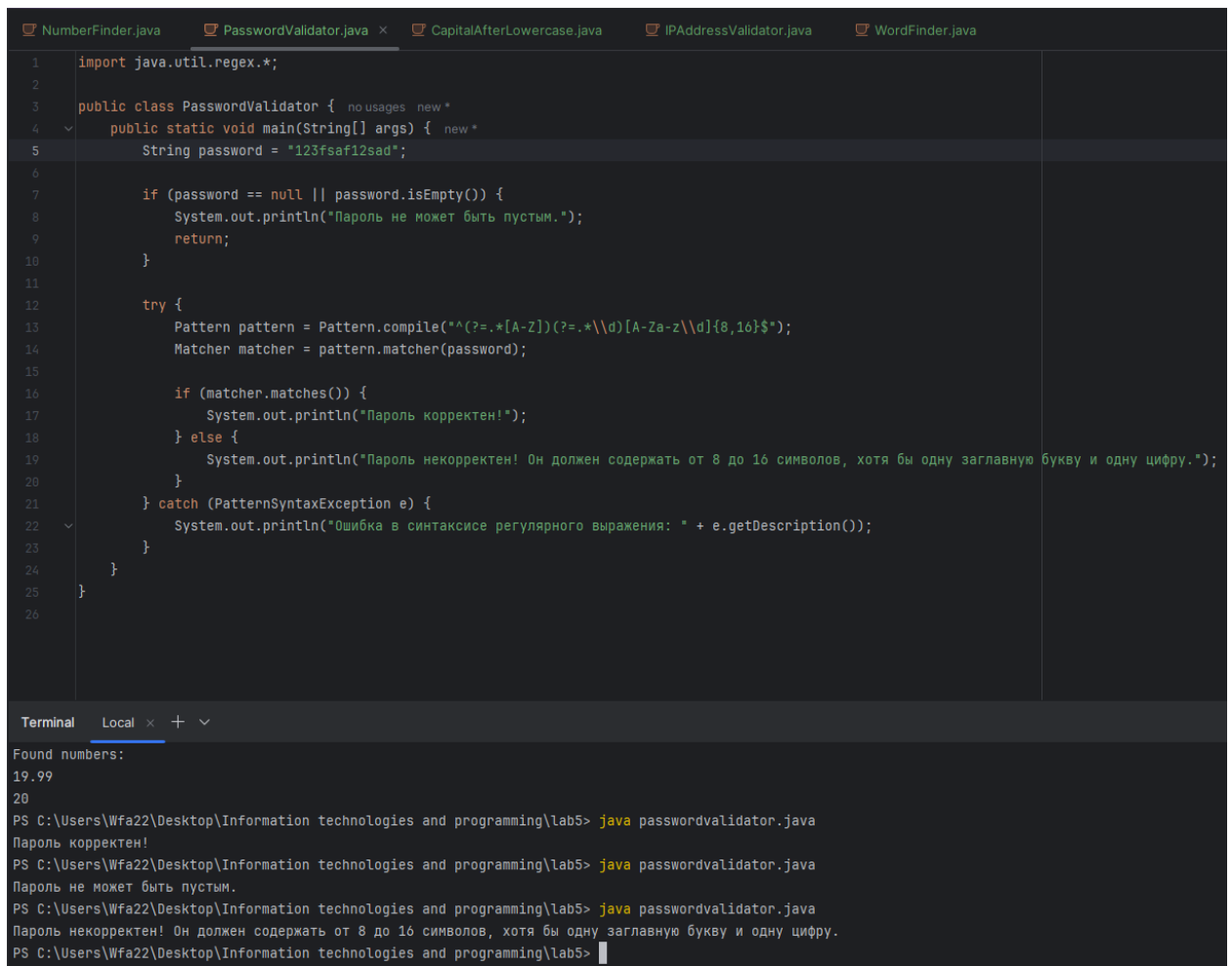
20

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java

Пароль корректен!

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java

Пароль не может быть пустым.



```
NumberFinder.java PasswordValidator.java CapitalAfterLowercase.java IPAddressValidator.java WordFinder.java
1 import java.util.regex.*;
2
3 public class PasswordValidator { no usages new *
4     public static void main(String[] args) { new *
5         String password = "123fsaf12sad";
6
7         if (password == null || password.isEmpty()) {
8             System.out.println("Пароль не может быть пустым.");
9             return;
10        }
11
12        try {
13            Pattern pattern = Pattern.compile("(?=[A-Z])(?=.*\\d)[A-Za-z\\d]{8,16}$");
14            Matcher matcher = pattern.matcher(password);
15
16            if (matcher.matches()) {
17                System.out.println("Пароль корректен!");
18            } else {
19                System.out.println("Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру.");
20            }
21        } catch (PatternSyntaxException e) {
22            System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
23        }
24    }
25 }
26
Terminal Local x + v
Found numbers:
19.99
20
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль корректен!
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль не может быть пустым.
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру.
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> 
```

Приступим к выполнению третьего задания:

По аналогии напишем наш код с уже проверками на пустую строку и синтаксическую ошибку, также добавим проверку на то, произойдет исключение во время замены, например, если `StringBuffer` вдруг окажется недоступен.

```
NumberFinder.java PasswordValidator.java CapitalAfterLowercase.java x IPAddressValidator.java WordFinder.java
1 import java.util.regex.*;
2
3 public class CapitalAfterLowercase { no usages new *
4     public static void main(String[] args) { new *
5         String text = "This is an example AbcDefGhi of finding patterns.";
6
7         if (text == null || text.isEmpty()) {
8             System.out.println("Текст не может быть пустым.");
9             return;
10        }
11
12        try {
13            Pattern pattern = Pattern.compile("([a-z])([A-Z])");
14            Matcher matcher = pattern.matcher(text);
15
16            StringBuffer result = new StringBuffer();
17            while (matcher.find()) {
18                matcher.appendReplacement(result, matcher.group(1) + "!" + matcher.group(2) + "!");
19            }
20            matcher.appendTail(result);
21
22            System.out.println("Результат: " + result.toString());
23        } catch (PatternSyntaxException e) {
24            System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
25        } catch (IllegalStateException e) {
26            System.out.println("Ошибка при замене текста: " + e.getMessage());
27        }
28    }
29 }
30
```

Terminal Local x + v

```
20
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль корректен!
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль не может быть пустым.
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java passwordvalidator.java
Пароль некорректен! Он должен содержать от 8 до 16 символов, хотя бы одну заглавную букву и одну цифру.
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java capitalafterlowercase.java
Результат: This is an example Abc!D!ef!G!hi of finding patterns.
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5>
```

Наше исключение `IllegalStateException` срабатывает тогда, когда например, попытаются вызвать `matcher.group()` до `matcher.find()`

Далее выполним четвертое задание:

Добавим проверки на синтаксис и пустое поле


```
NumberFinder.java PasswordValidator.java CapitalAfterLowercase.java IPAddressValidator.java x WordFinder.java
1 import java.util.regex.*;
2
3 public class IPAddressValidator { no usages new *
4     public static void main(String[] args) { new *
5         String ip = "192.168.1.1";
6
7         try {
8             if (ip == null || ip.isEmpty()) {
9                 System.out.println("IP-адрес не может быть пустым или null.");
10                return;
11            }
12
13            String ipPattern = "^((25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)\\.){3}(25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)$";
14            Pattern pattern = Pattern.compile(ipPattern);
15            Matcher matcher = pattern.matcher(ip);
16
17            if (matcher.matches()) {
18                System.out.println("IP-адрес корректен!");
19            } else {
20                System.out.println("IP-адрес некорректен!");
21            }
22        } catch (PatternSyntaxException e) {
23            System.out.println("Ошибка в синтаксисе регулярного выражения: " + e.getDescription());
24        } catch (NullPointerException e) {
25            System.out.println("Ошибка: IP-адрес не может быть null.");
26        }
27    }
28 }
29
```

Terminal Local x + v

Ошибка при замене текста: No match found

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java ipaddressvalidator.java

IP-адрес корректен!

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java ipaddressvalidator.java

IP-адрес корректен!

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java ipaddressvalidator.java

IP-адрес некорректен!

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java ipaddressvalidator.java

IP-адрес корректен!

PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> |

Подробное объяснение регулярного выражения

Разберём регулярное выражение `^((25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)\\.){3}(25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)$`:

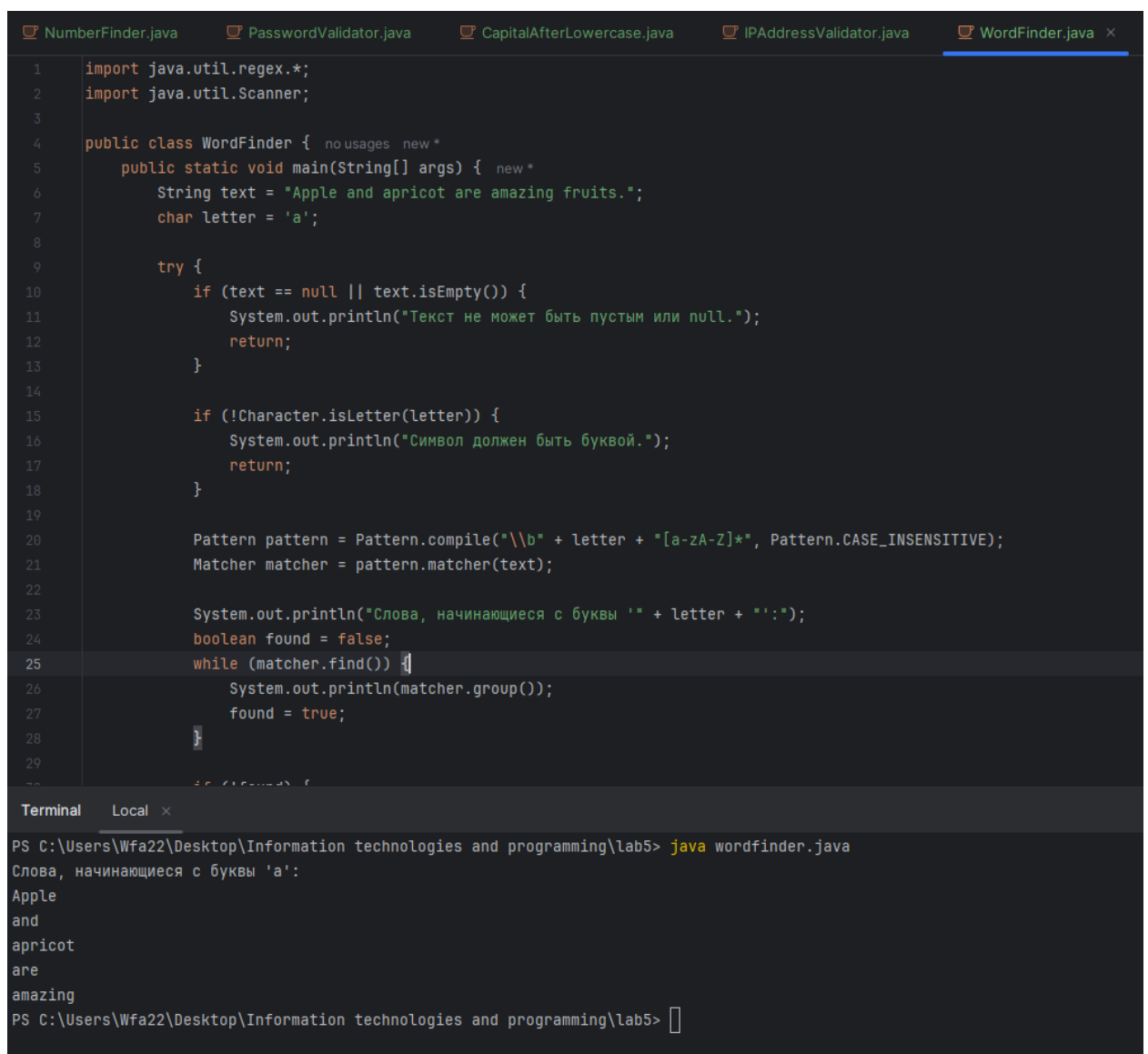
- `^` — начало строки. Указывает, что IP-адрес должен начинаться сразу с первого символа.
- `((25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)\\.){3}` — шаблон для первых трёх чисел в IP-адресе:
 - `25[0-5]` — число от 250 до 255.
 - `2[0-4]\\d` — число от 200 до 249.
 - `1\\d{2}` — число от 100 до 199.
 - `[1-9]?\\d` — одно- или двухзначное число от 0 до 99.
 - `\\.` — точка, разделяющая части IP-адреса.
 - `{3}` — указывает, что такая группа (число + точка) повторяется ровно 3 раза.
- `(25[0-5]|2[0-4]\\d|1\\d{2}|[1-9]?\\d)` — последняя группа числа (без точки):

- Аналогично предыдущей группе, это число должно быть от 0 до 255.
- \$ — конец строки, указывает, что IP-адрес заканчивается сразу после последнего числа.

Задание 5:

Напишем код и добавим проверки на пустой текст, некорректный символ для поиска, ошибку синтаксиса:

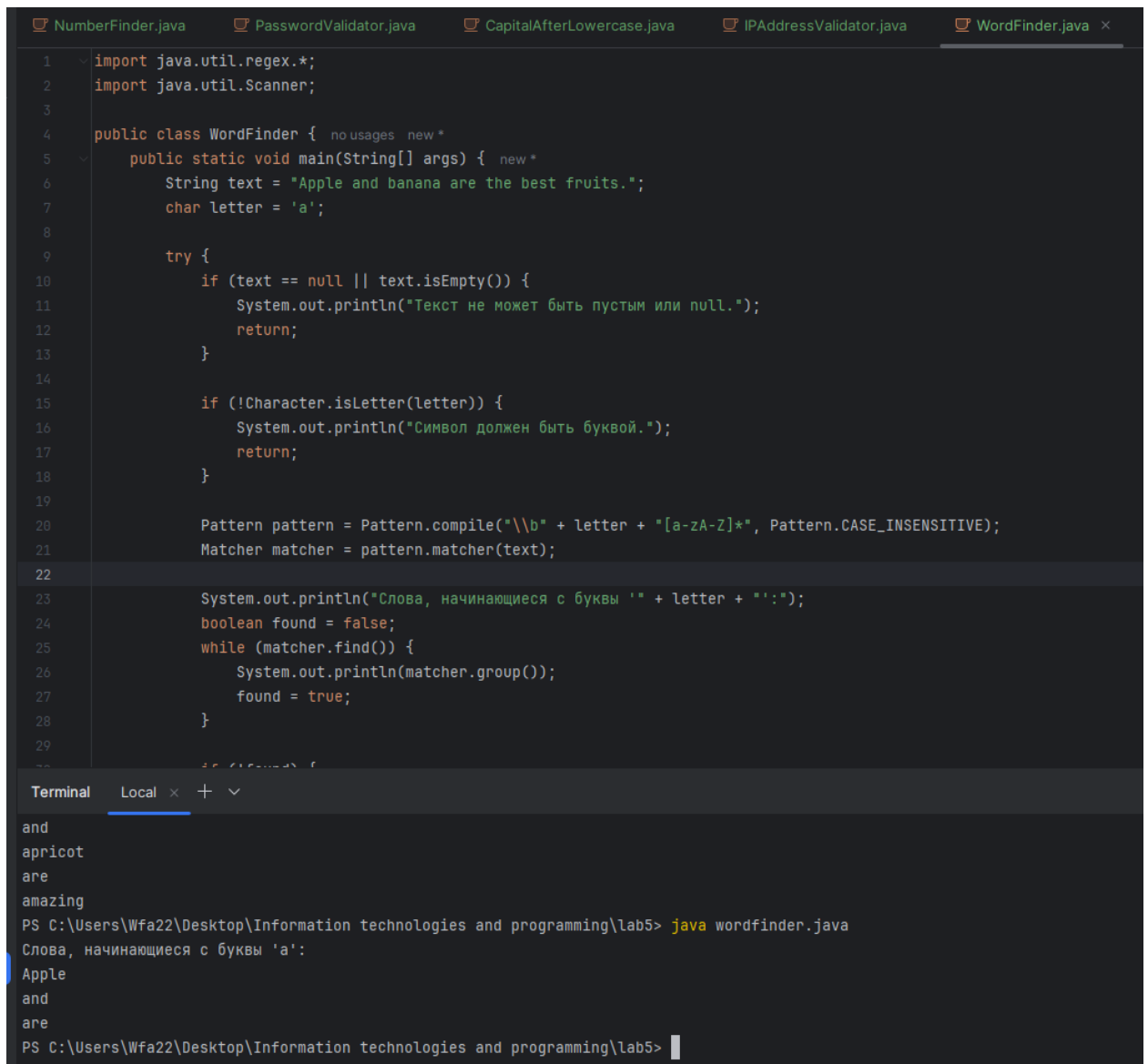
Объяснение регулярного выражения



```
NumberFinder.java PasswordValidator.java CapitalAfterLowercase.java IPAddressValidator.java WordFinder.java x
1 import java.util.regex.*;
2 import java.util.Scanner;
3
4 public class WordFinder { no usages new *
5     public static void main(String[] args) { new *
6         String text = "Apple and apricot are amazing fruits.";
7         char letter = 'a';
8
9         try {
10             if (text == null || text.isEmpty()) {
11                 System.out.println("Текст не может быть пустым или null.");
12                 return;
13             }
14
15             if (!Character.isLetter(letter)) {
16                 System.out.println("Символ должен быть буквой.");
17                 return;
18             }
19
20             Pattern pattern = Pattern.compile("\\b" + letter + "[a-zA-Z]*", Pattern.CASE_INSENSITIVE);
21             Matcher matcher = pattern.matcher(text);
22
23             System.out.println("Слова, начинающиеся с буквы '" + letter + "':");
24             boolean found = false;
25             while (matcher.find()) {
26                 System.out.println(matcher.group());
27                 found = true;
28             }
29
30             if (!found) {
31                 System.out.println("Слова не найдены.");
32             }
33         } catch (Exception e) {
34             System.out.println("Ошибка: " + e.getMessage());
35         }
36     }
37 }
```

Terminal Local x

```
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java wordfinder.java
Слова, начинающиеся с буквы 'a':
Apple
and
apricot
are
amazing
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> 
```



```
NumberFinder.java PasswordValidator.java CapitalAfterLowercase.java IPAddressValidator.java WordFinder.java x
1 import java.util.regex.*;
2 import java.util.Scanner;
3
4 public class WordFinder { no usages new *
5     public static void main(String[] args) { new *
6         String text = "Apple and banana are the best fruits.";
7         char letter = 'a';
8
9         try {
10             if (text == null || text.isEmpty()) {
11                 System.out.println("Текст не может быть пустым или null.");
12                 return;
13             }
14
15             if (!Character.isLetter(letter)) {
16                 System.out.println("Символ должен быть буквой.");
17                 return;
18             }
19
20             Pattern pattern = Pattern.compile("\\b" + letter + "[a-zA-Z]*", Pattern.CASE_INSENSITIVE);
21             Matcher matcher = pattern.matcher(text);
22
23             System.out.println("Слова, начинающиеся с буквы '" + letter + "':");
24             boolean found = false;
25             while (matcher.find()) {
26                 System.out.println(matcher.group());
27                 found = true;
28             }
29         } catch (Exception e) {
30             e.printStackTrace();
31         }
32     }
33 }
```

Terminal Local x + v

```
and
apricot
are
amazing
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5> java wordfinder.java
Слова, начинающиеся с буквы 'a':
Apple
and
are
PS C:\Users\Wfa22\Desktop\Information technologies and programming\lab5>
```

Регулярное выражение `\\b" + letter + "[a-zA-Z]*"` использует несколько важных компонентов для поиска слов, начинающихся с заданной буквы:

- `\\b` — граница слова. Эта конструкция указывает на начало слова, чтобы убедиться, что искомая буква является началом слова.
- `+ letter` — добавляет букву, которую ввел пользователь. Это значение объединяется с остальной частью регулярного выражения, чтобы задать начальный символ искомого слова.
- `[a-zA-Z]*` — указывает, что после начальной буквы могут следовать ноль или более букв. Диапазон `[a-zA-Z]` включает в себя все латинские буквы, независимо от регистра.
- `Pattern.CASE_INSENSITIVE` — делает поиск нечувствительным к регистру, чтобы находить как заглавные, так и строчные варианты слов, начинающихся с заданной буквы.

Принцип работы регулярного выражения

1. Регулярное выражение находит начало слова с помощью `\\b`.
2. Далее оно проверяет, что слово начинается с заданной буквы `letter`.

3. После первой буквы могут идти другие буквы (или их может не быть вообще), что позволяет захватить всё слово.

Вывод: изучили теорию по регулярным выражениям, выполнили задания.