Pattern и Matcher в Java:

Pattern: Класс Pattern в Java используется для создания шаблона регулярного выражения. Он компилирует представление регулярного выражения в формат, который можно использовать для поиска, сопоставления и манипуляций с текстом.

Matcher: Класс Matcher интерпретирует шаблон и выполняет операции сопоставления с текстовыми последовательностями. Он работает вместе с классом Pattern.

Алгоритм поиска в Matcher - Жадный и Ленивый:

По умолчанию, алгоритм поиска в Matcher жадный, что означает, что он пытается сопоставить как можно больше текста.

Ленивый поиск можно добиться, добавив ? после квантификатора (например, *?, +?, {n,m}?).

PatternSyntaxException:

PatternSyntaxException — это исключение, которое выбрасывается при синтаксической ошибке в регулярном выражении. Это исключение содержит информацию о шаблоне, описание ошибки и индекс в шаблоне, где произошла ошибка.

Символ . в регулярных выражениях:

В регулярных выражениях . означает любой символ (кроме перевода строки).

Для поиска по символу точки ., его необходимо экранировать: \\..

Поиск по символу \ также требует экранирования: \\\\.

Экранирование применяется, когда нужно искать символы, имеющие специальное значение в регулярных выражениях.

Символы *, +, ?, {3,5}:

*: Соответствует предыдущему элементу ноль или более раз.

+: Соответствует предыдущему элементу один или более раз.

?: Соответствует предыдущему элементу ноль или один раз.

{3,5}: Соответствует предыдущему элементу минимум 3, максимум 5 раз.

Проверка совпадения всей строки с регулярным выражением:

Чтобы проверить, совпадает ли вся строка с регулярным выражением, используйте matches() метод класса Matcher.

Исключение символов из поиска:

Чтобы исключить определенные символы из поиска, используйте отрицательные классы символов, например [^abc] исключит символы 'a', 'b', 'c'.

Именованные группы в регулярных выражениях:

Именованная группа создается с помощью синтаксиса (?<name>X), где name — имя группы, а X — регулярное выражение.

В примере ((A(B))(C)), группа 1 это (A(B)), а группа 3 это C.

Преимущество StringBuilder перед String:

StringBuilder эффективнее для операций изменения строк, так как он избегает создания новых строковых объектов при каждом изменении. В отличие от String, который является неизменяемым и каждая операция с ним создаёт новый объект строки, StringBuilder позволяет изменять строку на месте, что уменьшает накладные расходы на выделение памяти и увеличивает производительность при частых операциях со строками.

Регулярное выражение для поиска слов на 'C' и заканчивающихся на 't' или 'e':

Если нам нужно регулярное выражение, которое соответствует словам, начинающимся на 'C' и заканчивающимся на 't' или 'e'. Это может быть выражено как C\w*e|C\w*t. Здесь \w* соответствует любому количеству буквенных символов.

Нахождение позиций небуквенных символов в строке:

Чтобы найти позиции всех небуквенных символов в строке, можно использовать регулярное выражение [^A-Za-z], где ^ внутри квадратных скобок обозначает отрицание, то есть соответствие любому символу, кроме буквенного.

Проверка строки на соответствие двоичному представлению:

Чтобы проверить, является ли строка двоичным числом (содержит только '1' и '0'), можно использовать регулярное выражение ^[01]+$. Если строка соответствует этому выражению, она является двоичным числом.

Если в строке есть символы, отличные от '0' и '1', можно найти их позиции, используя отрицательный класс символов [^01].