Taller Expresiones Regulares

Estudiantes

- Sebastián López Osorno

Instrucciones

La solución debe ser enviada al correo electrónico: alvaromontoya@elpoli.edu.co
Tendrá un valor de una unidad en la nota de la segunda práctica (20% de la nota).

Preguntas

¿Qué es un patrón?

Un patrón, en el contexto de las expresiones regulares, es una secuencia definida de caracteres que se utiliza para buscar coincidencias en cadenas de texto. Estos patrones pueden incluir caracteres literales, metacaracteres, y cuantificadores que permiten describir y coincidir con complejas estructuras y dinámicas dentro del texto.

Explique, ¿por qué las siguientes series siguen un patrón?

Serie: `1 1 2 1 2 3 1 2 3 4 1 2 3 4 5…`

Esta serie sigue un patrón porque cada bloque de números empieza en 1 y se incrementa hasta un número que va aumentando con cada repetición. La longitud de cada secuencia es igual al número máximo que alcanza.

Serie de Fibonacci: `0 1 1 2 3 5 8 13 21…`

Esta es una serie de Fibonacci, donde cada número es la suma de los dos anteriores. Este patrón de suma recursiva forma la base de la serie.

Serie: `2 7 4 14 6 21 8 28 10 35 12 42 14 49 16 56…`

Esta serie sigue un patrón donde los números impares son progresiones aritméticas y los números pares son múltiplos de siete. Además, cada número par después del primero se forma sumando siete al número anterior de su tipo.

Cree una serie con un patrón que incluya letras y números (explique)

Serie: `A1 B2 C3 D4 E5 F6 …` **Explicación**: En esta serie, cada letra del alfabeto es seguida por un número que representa su posición en el alfabeto. A1 es la primera letra y el número 1, B2 es la segunda letra y el número 2, y así sucesivamente.

Cree una serie con un patrón que incluya solo letras (Explique)

Serie: `A B C D E F G …` **Explicación**: Esta serie muestra un patrón simple donde cada letra sigue alfabéticamente a la anterior. Comienza con A y continúa secuencialmente a través del alfabeto.

En programación de software, ¿qué son expresiones regulares?

Las expresiones regulares son una herramienta poderosa en programación para la búsqueda y manipulación de textos. Permiten definir patrones de búsqueda complejos que pueden incluir variaciones y condiciones específicas para encontrar coincidencias dentro de grandes volúmenes de texto de manera eficiente y efectiva.

¿En qué casos sirven las expresiones regulares?

Las expresiones regulares son extremadamente útiles en varios escenarios como:

- **Validación de datos**: Asegurar que los datos ingresados por los usuarios siguen un formato específico, como correos electrónicos, números de teléfono, o códigos postales.

- **Búsqueda de patrones**: Extraer información específica de textos, como fechas, nombres, o códigos específicos dentro de documentos o flujos de datos.
- **Manipulación de texto**: Modificar textos de acuerdo a patrones específicos, como eliminar espacios innecesarios, cambiar formatos de fecha, o anonimizar datos sensibles en textos.

Normas de las expresiones regulares

Las expresiones regulares operan bajo un conjunto de normas que definen cómo se interpretan los patrones:

- **Metacaracteres**: Caracteres especiales que realizan funciones específicas, como `.` (cualquier carácter), `*` (cero o más del carácter anterior), y `+` (uno o más del carácter anterior).
- **Cuantificadores**: Definen cuántas veces debe aparecer un elemento para cumplir con el patrón.
- **Conjuntos de caracteres**: Permiten especificar un grupo de caracteres entre los cuales puede haber una coincidencia, como `[a-z]` para cualquier letra minúscula.
- **Anclas**: Especifican la posición en el texto donde debe ocurrir una coincidencia, como `^` para el inicio de una línea o `$` para el final.
- **Grupos de captura**: Permiten agrupar parte de una expresión regular y recuperar lo que coincide con esa parte del patrón.

Ejemplos de expresiones regulares

1. **Expresión Regular**: `\d{3}-\d{2}-\d{4}` **Descripción**: Valida números de Seguro Social en formato estadounidense, consistiendo en tres dígitos, seguido de un guion, dos dígitos más, otro guion, y cuatro dígitos al final.

2. **Expresión Regular**: `^[a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]{2,6}$` **Descripción**: Verifica que una cadena tenga el formato de un correo electrónico, con partes consistiendo en caracteres alfabéticos, seguido de `@`, más caracteres alfabéticos, un punto y una extensión de dominio de entre 2 a 6 letras.

3. **Expresión Regular**: `^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)[a-zA-Z\d]{8,}$` **Descripción**: Asegura que las contraseñas contengan al menos 8 caracteres, incluyendo una letra minúscula, una mayúscula y un número, utilizando

afirmaciones positivas adelantadas para verificar cada condición sin capturar los caracteres.

Paquete de Java para expresiones regulares

El paquete `java.util.regex` es el proporcionado por Java para trabajar con expresiones regulares. Contiene clases como `Pattern` y `Matcher` que permiten compilar y ejecutar expresiones regulares contra cadenas de texto.

¿Para qué se utiliza la clase `Matcher`?

La clase `Matcher`, parte del paquete `java.util.regex`, se utiliza para interpretar patrones contra secuencias de caracteres y realizar operaciones de búsqueda, extracción, y manipulación de texto. Proporciona métodos como `find()`, `matches()`, y `group()` que son esenciales para trabajar con coincidencias detalladas dentro de cadenas.

Ejemplo de uso de `Matcher`

```
Pattern pattern = Pattern.compile("a*b");
Matcher matcher = pattern.matcher("aabmanoloaabmanoloabmanolob");
while (matcher.find()) {
    System.out.println("Coincidencia encontrada: " + matcher.group());
}
// Este código busca todas las secuencias que consisten en cero o más 'a' seguidas
por un 'b'.

### ¿Para qué se utiliza el método `pattern`?
El método `pattern` se utiliza para compilar una expresión regular en un objeto
`Pattern` en Java. Este objeto se puede utilizar para crear un `Matcher` que
realizará las operaciones de coincidencia en texto.

#### Ejemplo de uso de `pattern`
```java
    Pattern pattern = Pattern.compile("\\d{3}-\\d{2}-\\d{4}");
    Matcher matcher = pattern.matcher("123-45-6789");
    if (matcher.matches()) {
        System.out.println("Formato válido.");
    }
```

// Este código verifica si una cadena cumple con el formato de un número de Seguro Social.
```

¿Para qué se utiliza el método `matcher`?

El método `matcher` se utiliza para crear un objeto `Matcher` de una cadena dada, que permite realizar múltiples operaciones de coincidencia y extracción basadas en el patrón compilado.

Ejemplo de uso de `matcher`

```
String input = "hello, world!";
Pattern pattern = Pattern.compile("\\w+");
Matcher matcher = pattern.matcher(input);
while (matcher.find()) {
    System.out.println(matcher.group());
}
```

// Este código encuentra y muestra todas las palabras en la cadena de entrada.

¿Para qué se utiliza el método `split`?

El método `split` de la clase `String` se utiliza para dividir una cadena en un array de subcadenas basadas en un delimitador especificado que puede ser una expresión regular.

Ejemplo de uso de `split`

```
public class SplitExample {
    public static void main(String[] args) {
        String texto = "Uno, Dos, Tres, Cuatro";
        String[] resultados = texto.split(", ");

        for (String resultado : resultados) {
            System.out.println(resultado);
        }
    }
}
```

¿Para qué se utiliza el método `matches`?

El método `matches` se utiliza para verificar si una cadena completa coincide con el patrón especificado en la expresión regular. Devuelve `true` si la cadena coincide completamente y `false` en caso contrario.

Ejemplo de uso de `matches` y diferencia con `matcher`

```
public class MatchesExample {
    public static void main(String[] args) {
        String texto = "12345";
        boolean resultado = texto.matches("\\d{5}");

        System.out.println("La cadena es un número de cinco dígitos: " +
resultado);
    }
}
```

¿Qué cadena de texto imprime los siguientes procedimientos?

Ejemplo 1: Uso de `Matcher` y `replaceAll`

```
import java.util.regex.*;

public class Ejemplo {
    public static void main(String args[]) {
        Pattern patron = Pattern.compile("a*b");
        Matcher encaja = patron.matcher("aabmanoloaabmanoloabmanolob");
        System.out.println(encaja.replaceAll("-"));
    }
}
```

Este procedimiento reemplaza todas las ocurrencias de cualquier número de 'a' seguido por un 'b' con un guión. La salida será: -manolo-manolo-manolo.

Ejemplo 2: Método matches para verificar valores booleanos

```
public boolean method(String s) {
    return s.matches("[tT]rue|[yY]es");
```

```
}
```

Este método devuelve true si s es exactamente "True", "true", "Yes", o "yes". Devuelve false para cualquier otra entrada.

Ejemplo 3: Método matches para encontrar 'true' en cualquier posición

```
public boolean method(String s) {
    return s.matches(".*true.*");
}
```

Este método devuelve true si la palabra "true" aparece en cualquier parte de la cadena s, sin importar otros caracteres antes o después.

Ejemplo 4: Método matches para validar tres letras

```
public boolean method(String s) {
    return s.matches("[a-zA-Z]{3}");
}
```

Este método devuelve true si la cadena s está compuesta exactamente por tres letras (mayúsculas o minúsculas).

Ejemplo 5: Método matches para iniciar sin dígito

```
public boolean method(String s) {
    return s.matches("[^\\d].*");
}
```

Este método devuelve true si la cadena s inicia con cualquier carácter que no sea un dígito.

Ejemplo 6: Método matches para caracteres alfanuméricos sin 'b'

```
public boolean method(String s) {
    return s.matches("([\\w&&[^b]])*");
}
```

Este método devuelve true si la cadena s contiene solo caracteres alfanuméricos y ninguno de ellos es la letra 'b'.

Ejemplo 7: Método matches para verificar una cadena numérica opcional

```
public boolean method(String s) {
    return s.matches("[1,2]?[0-9]{1,2}");
}
```

Este método devuelve true si la cadena s consta de uno a tres dígitos numéricos, donde el primer dígito puede ser opcionalmente '1' o '2'.

Ejemplo 8: Procesamiento de entrada de usuario en tiempo real

```
import java.util.Scanner;

public class PracticaExpresiones {
    private static Scanner leer = new Scanner(System.in);
    public static void main(String args[]) {
        System.out.println("INGRESE TEXTO: ");
        String texto = leer.nextLine();
        if (texto.matches("^([a-zA-Z]+[ ]*)*$")) {
            char[] cadenaTexto = texto.toCharArray();
            for (int i = 0; i < cadenaTexto.length; i++) {
                char aux = cadenaTexto[i];
                if (aux != ' ') {
                    aux++;
                    System.out.print(aux);
                } else {
                    System.out.print(aux);
```

```
                }
            }
        }
    }
}
```

Este programa solicita al usuario ingresar texto y luego incrementa cada carácter en una posición en el alfabeto si el texto solo contiene letras y espacios. Por ejemplo, el texto "Abc" se convertirá en "Bcd".