Expresiones Regulares

Marthel Pedro Pozo Soliz

13 de julio de 2024

1. Introducción

Las expresiones regulares o regex (también regexp) son una herramienta en el ámbito de la informática y programación que nos permite describir patrones de texto de manera concisa y eficiente, su uso se extiende a tareas como: Validación de entradas, búsqueda y reemplazo de textos, análisis de sintaxis y mas. Las expresiones regulares están basadas en la teoría de lenguajes formales y autómatas, y son fundamentales en la programación y el procesamiento de texto.

2. Definición de Expresiones Regulares

Una Expresión regular en una secuencia de caracteres que define un patrón de búsqueda. Se utilizan ampliamente en programación para manipular, buscar y validar textos de manera eficiente. Nos permiten realizar búsquedas complejas y flexibles, como encontrar coincidencias que sigan ciertos criterios específicos, reemplazar y/o extraer porciones de texto que cumplan con un patrón determinado.

3. Características de las Expresiones Regulares

3.1. Ventajas

- Flexibilidad: Se adaptan una gran variedad de necesidades de manipulación de texto.
- Eficiencia: Tienen un funcionamiento optimizado que proviene de su propia implementación en los diferentes lenguajes de programación y entornos.
- Concisión: Se pueden representar patrones complejos de manera compacta.
- Compatibilidad: La mayoría de lenguajes de programación tienen implementados las expresiones regulares con un estándar común.

3.2. Desventajas

- Complejidad: Algunos patrones al ser demasiado complejos suelen ser difíciles de mantener o interpretar por un humano.
- Limitación: No son adecuadas para todas las tareas de procesamiento.

3.3. Aplicaciones

- **Validación de entradas:** Se usan expresiones regulares para validar formatos de entrada de texto especificados por el programador.
- **Búsqueda y Reemplazo de Texto:** Gran variedad de editores de texto, código tienen expresiones regulares implementadas en sus búsquedas
- **Scrapping:** Se usan activamente para la recolección de datos en paginas web, documentos, o cualquier formato que tenga datos no organizados.
- **Compiladores e interpretes:** Utilizadas en la fase de análisis léxico para dividir el código fuente en tokens.

4. Elementos de las Expresiones Regulares

Las expresiones regulares tienen elementos que se usan para el procesamiento del texto. Entre ellos:

- **Caracteres literales:** Se corresponden exactamente con el texto que representan.

```
'a' reconoce {'a'}
```

- **Meta-caracteres:** Son caracteres de control cada uno con un significado propio.
  - **Diagonal Invertida (\):** Escape de clase carácter, indica que el siguiente carácter debe usarse como carácter en lugar de meta-carácter.

```
"\*" = "*"
```

  - **Punto(.):** Reconoce cualquier carácter individual.

```
"." reconoce {'a', 'b', 'c', 'd', 'e', ...}
```

  - **Caret ($\hat{}$):** Reconoce un texto que inicia con el cadena que sigue a la expresión.

```
"^hola" reconoce {"hola mundo", "hola, juan", "hola ...",
    etc}
```

  - **Dólar ($):** Reconoce un texto que termina con la cadena que precede a la expresión

```
"hola$" reconoce {"mundo hola", "juan, hola", "... hola",
    etc}
```

- **Interrogación (?):** Reconoce carácter o grupo anterior a la expresión cero o una vez.

```
"(hol)?a" reconoce {"a", "hola"}
```

- **Asterisco (*):** Reconoce el carácter o grupo anterior a la expresión cero o mas veces.

```
"(hol)*a" reconoce {"a", "hola", "holhola", etc}
```

- **Signo mas (+):** Reconoce el carácter o grupo anterior a la expresión una o mas veces.

```
"(hol)+a" reconoce {"hola", "holhola", etc}
```

- **Barra vertical (|):** Reconoce el carácter o grupo anterior o el carácter o grupo siguiente de la expresión.

```
"a|b" reconoce {'a', 'b'}
```

- **Grupos:** Se usan para representar la agrupación de caracteres con cierto significado común.
  - **Corchetes ([,]):** Definen una **clase carácter** compuesta por los Corchetes y el contenido dentro de ellos.

```
[aeiuo] reconoce cualquier vocal
        {'a','e','i','o','u'}
```

En adición es posible usar rangos de caracteres

```
[a-z] reconoce cualquier letra minuscula.
        {'a','b','c',...,'z'}

[a-p] reconoce cualquier letra minuscula de la a a la p.
        {'a','b','c',...,'p'}

[0-9] reconoce cualquier digito unico.
        {0,1,2,3,4,5,6,7,8,9}
```

- **Paréntesis (‘(‘ , ‘)‘):** Representa una agrupación de caracteres o expresiones a los que se les puede aplicar cualquier expresión

```
([aeiuo])+ reconoce cualquier combinacion de vocales
        {"aa", "aeiuo", "aaae", "aaeeiiuuoo"}

(hol)+a reconoce infintas "hol" concatenada a una "a"
        {"hola", "holhola", etc}

(ab|cd)+ef reconoce uno o mas "ab" o "cd" concatenados con
     "ef"
        {"abef", "cdef", "ababef", "abcdabef", etc}
```

- **Escape Clase Carácter:** Un escape de clase de carácter especifica que determinados caracteres especiales deben tratarse como caracteres, en lugar de realizar alguna función. Se indica su uso con una barra invertida (backslash) \ seguida del metacaracter a representar.

Escape Caracter	Nombre
\n	Nueva linea
\t	Tabulador
\r	Retorno
\\ |	Barra Invertida (backslash)
\^ ^	Acento Circunflejo (Caret)
\. .	Punto (Periodo)
\? ?	Interrogante
* *	Asterisco
\+ +	Signo adicion
\((	Parentesis de apertura
\))	Parentesis de cierre
\{ {	Llave de apertura
\} }	Llave de cierre
\$ $	Signo de dolar
\- -	Guion
\[[	Corchete de apertura
\]]	Corchete de cierre

- **Cuantificadores:** Son operadores que determinan cuántas veces debe aparecer un patrón para que coincida con una cadena de texto.

Simbolo	Nombre
*	Cero o mas
+	Uno o mas
?	Cero o uno
{n}	n Veces
{n,m}	Entre n o m Veces
{n,}	n o mas Veces

- **Otros metacaracteres predefinidos:**

Simbolo	Metacaracter	Descripcion

```
\d      Digitos          Cualquier digito del 0 al 9.

\D      No Digitos       Cualquier caracter que no
                         sea un digito.

\w      Palabra          Coincide con letras, digitos y
                         el guion bajo (_).

\W      No Palabra       Coincide con cualquier caracter que no
                         sea una letra, digito o guion bajo.

\s      Espacio          Coincide con cualquier caracter de
                         espacio en blanco ( , \t, \n, \r, etc)

\S      No Espacio       Coincide con cualquier caracter que no
                         sea un espacio en blanco.

\b      Limite de Palabra       Coincide con el borde de
                         una palabra.

\B      No Limite de Palabra    Coincide con cualquier
                         lugar que no sea un limite de palabra.
```

5. Expresiones Regulares en Java

Para trabajar con expresiones regulares en **Java** podemos usar las clases **Pattern** y **Matcher** que vienen incluidas en el paquete ***java.util.regex***. La clase **Pattern** tendrá el patrón (o expresión regular) a usar y la clase **Matcher** contiene el patrón (o expresión regular) y el texto en la cual se aplicara la expresión regular.

```java
// importar ls clases Matcher y Pattern
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {
    public static void main(String[] args) {
        //Texto a procesar
        String texto = "hola mundo";
        //Expresion Regular
        Pattern patron = Pattern.compile("^hola.*");
        //Procesado expresion-texto
        Matcher coincidencia = patron.matcher(texto);
        //Busqueda
        if (coincidencia.find()) {
            System.out.println(coincidencia.group());
        }
    }
}

// reconoce {"hola", "hola mundo", "hola, que tal", "hola..."}, etc}
```

Métodos disponibles Clase Pattern

- compile(): Compila la expresión en un patrón
- matcher(): Crea un Matcher, este puede buscar el patrón en un texto.

- split(String input): Divide el texto (input) en un array separandolo por el patrón.

Clase Matcher

- find(): Busca la siguiente coincidencia.
- group(): Retorna la coincidencia.
- replaceAll(String reemplazo): Reemplaza todas las coincidencias con el texto (reemplazo).
- replaceFirst(String reemplazo): Reemplaza la primera coincidencia con el texto (reemplazo).
- matches(): Verifica si toda la cadena coincide con el patrón.

6. Ejemplos Prácticos

6.1. Obtencion de fechas

Ejemplo de obtencion de multiples fechas de un array de strings

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Fechas {

    public static void main(String[] args) {
        String[] diario={
            "hoy 1824-06-02 encontre una solucion al problema",
            "1824-06-06 inicie la busqueda de la planta",
            "mañana 1824-06-24 termina la expedicion",
            "el dia de hoy 1824-07-12 iniciamos el retorno"
        };
        Pattern fecha = Pattern.compile("\\d{4}-\\d{2}-\\d{2}");
        System.out.println("Fechas: ");
        for (String dia : diario) {
            Matcher fechas = fecha.matcher(dia);
            if(fechas.find())
                System.out.println(fechas.group());
        }
    }
}
```

Se busca una cadena de la forma **dddd-dd-dd** siendo d un digito cualquiera
Expresiones usadas:

```
\d{4}   4 digitos
\d{2}   2 digitos
-       Guion Caracter literal
```

Resultados esperados:

```
Fechas:
        1824-06-02
        1824-06-06
        1824-06-24
        1824-07-12
```

6.2. Login

En este ejemplo se trata de verificar el correo electronico de una cuenta para proceder a un login

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import javax.swing.*;
import java.awt.*;

public class Login extends JFrame {
    private JTextField mailField;
    private JPasswordField passwordField;
    private JButton loginButton;
    Pattern mailFormat = Pattern.compile("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$");
    Matcher matcher;
    public static void main(String[] args) {
        new Login();
    }
    public Login() {
        init();
        eventHandler();
        loginButton.setEnabled(false);

    }
    public void eventHandler() {
        mailField.addKeyListener(new java.awt.event.KeyAdapter() {
            public void keyReleased(java.awt.event.KeyEvent evt) {
                matcher = mailFormat.matcher(mailField.getText());
                if(matcher.matches())
                    mailField.setBackground(Color.GREEN);
                else
                    mailField.setBackground(Color.RED);
                loginButton.setEnabled(matcher.matches());
            }
        });
    }
```

Se Verifica si un correo electronico comple con el formato **username@dominio.suf** Siendo username y dominio cadenas de caracteres, digitos, signos, y dominio una cadena de al menos dos letras.

Expresiones usadas:

^	Inicio de linea
[a-zA-Z0-9._%+-]+	Uno o mas caracteres (letras mayusculas o minusculas, digitos o simbolos [._%+-])
@	Caracter literal arroba @
[a-zA-Z0-9.-]+	Unos o mas caracteres (letras mayusculas o minusculas, digitos o simbolos [.-])
\.	Caracter literal punto .
[a-zA-Z]{2,}$	Dos o mas caracteres (letras mayusculas o minusculas)

Resultados Esperados:

El programa podra reconocer los correos electronicos ingresados en el JTextfield mailField, desactivara el boton Login si no reconoce un correo valido.

6.3. Identificar el IP local de una lista de IP's

Ejemplo de identificacion de direcciones IP locales en una cadena de texto usando expresiones regulares en Java. El objetivo es identificar direcciones IP locales en el formato '192.168.x.x'.

Se busca una cadena que contenga direcciones IP locales en el formato **192.168.x.x**, donde x puede ser cualquier número entre 0 y 255.

Expresiones usadas:

```
    192.168.    Prefijo para IP local en redes privadas
    .           Punto como separador
    \d{1,}      Uno o mas digitos
```

Código Java:

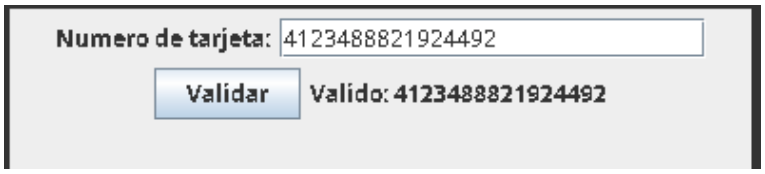
```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class IPLocal {
    public static void main(String[] args) {
        String ips="8.8.8.8 203.0.113.1 198.51.100.2 172.217.16.78
    93.184.216.34 192.0.2.3 192.168.43.203 151.101.1.69
    104.26.10.78 13.227.73.123";
        Pattern ipLocalFormat = Pattern.compile("192.168\\.\\d
    {1,}\\.\\d{1,}");
        Matcher ip = ipLocalFormat.matcher(ips);
        if(ip.find())
            System.out.println("IP local identificado:" + ip.group
    ());
    }
}
```

Resultados esperados:

```
IP local identificado: 192.168.43.203
```

6.4. Validación de números de tarjetas de crédito

Ejemplo de validación de números de tarjetas de crédito. La aplicación permite al usuario ingresar un número de tarjeta de crédito y verifica si es válido para Visa, MasterCard, o American Express.



Se busca una cadena que sea un número de tarjeta de crédito válido para Visa, MasterCard o American Express. Los patrones de expresión regular utilizados para cada tipo de tarjeta son los siguientes:

- **Visa**: Comienza con un 4 seguido de 12 o 15 dígitos.
- **MasterCard**: Comienza con un 5 seguido de un dígito del 1 al 5 y 14 dígitos.
- **American Express**: Comienza con 34 o 37 seguido de 13 dígitos.

Expresiones usadas:

```
^                               Inicio de cadena
(?:                             Inicia grupo
4[0-9]{12}(?:[0-9]{3})?         Valida Visa
|                               Union "or"
5[1-5][0-9]{14}                 Valida MasterCard
|                               Union "or"
 3[47][0-9]{13}                 Valida American Express
)                               Fin del grupo
$                                  Fin de cadena
```

Código Java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.regex.Pattern;

public class CardChecker extends JFrame {

    private JTextField cardNumberField;
    private JButton validateButton;
    private JLabel messageLabel;

    public CardChecker() {
        setTitle("Validar tarjetas visa/mastercard/american express
    ");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        cardNumberField = new JTextField(20);
        validateButton = new JButton("Validar");
        messageLabel = new JLabel("");

        validateButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String cardNumber = cardNumberField.getText().trim
    ();
                if (validateCardNumber(cardNumber)) {
                    messageLabel.setText("Valido:" + cardNumber);
                } else {
                    messageLabel.setText("Invalido.");
                }
            }
        });

        setLayout(new FlowLayout());
        add(new JLabel("Numero de tarjeta:"));
        add(cardNumberField);
        add(validateButton);
        add(messageLabel);
    }

    private boolean validateCardNumber(String cardNumber) {
        // Basic regex for credit card number validation
        Pattern pattern = Pattern.compile("
    ^(?:4[0-9]{12}(?:[0-9]{3})?|5[1-5][0-9]{14}|3[47][0-9]{13})$");
        return pattern.matcher(cardNumber).matches();
```

```
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new CardChecker().setVisible(true);
            }
        });
    }
}
```

Resultados esperados:

```
Numero de tarjeta: 4111111111111111
Valido: 4111111111111111

Numero de tarjeta: 5500000000000004
Valido: 5500000000000004

Numero de tarjeta: 378282246310005
Valido: 378282246310005

Numero de tarjeta: 4111111111111
Invalido.

Numero de tarjeta: 1234567890123456
Invalido.
```

6.5. Validación de nombres de archivo para la carga de imágenes

Ejemplo de validación de nombres de archivo en una aplicación de carga de imágenes usando `JFrame`. La aplicación permite al usuario seleccionar una imagen y verifica si el archivo es una imagen en formato `.jpg` o `.png`.

Se busca validar que el archivo seleccionado tenga una extensión `.jpg` o `.png`. La expresión regular utilizada asegura que el nombre del archivo termine en `.jpg` o `.png`, sin importar si las letras están en mayúsculas o minúsculas.

Expresiones usadas:

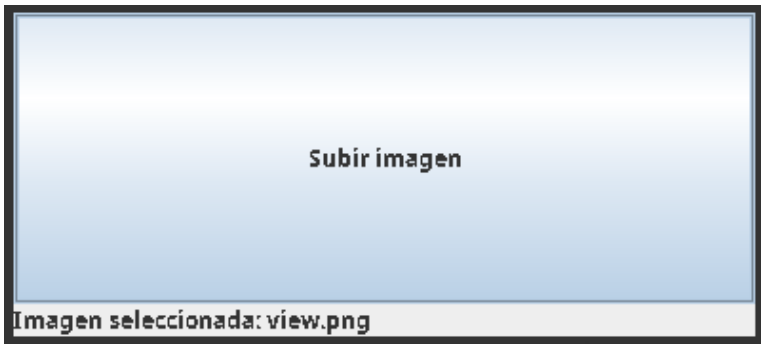
```
[^\s]+          Uno o mas caracteres que no sean espacios
(\.             Punto literal
(?i)            Indicador para hacer la busqueda sin importar
                    mayusculas/minusculas
(jpg|png)       Extensiones validas
)$              Fin de la cadena
```

Código Java:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;

public class ImageUpload extends JFrame {

    private JButton uploadButton;
    private JLabel messageLabel;

    public ImageUpload() {
        setTitle("Image Upload");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        uploadButton = new JButton("Subir imagen");
        messageLabel = new JLabel("Esperando archivo...");

        uploadButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                JFileChooser fileChooser = new JFileChooser();
                int result = fileChooser.showOpenDialog(null);
                if (result == JFileChooser.APPROVE_OPTION) {
                    File selectedFile = fileChooser.getSelectedFile
    ();
                    if (validateFileName(selectedFile.getName())) {
                        messageLabel.setText("Imagen seleccionada:
    " + selectedFile.getName());
                    } else {
                        messageLabel.setText("Archivo no valido,
    solo se permiten imagenes.");
                    }
                }
            }
        });

        setLayout(new BorderLayout());
        add(uploadButton, BorderLayout.CENTER);
        add(messageLabel, BorderLayout.SOUTH);
    }

    private boolean validateFileName(String fileName) {
        Pattern pattern = Pattern.compile("([^\\s]+(\\.(?i)(jpg|png
    ))$)");
        Matcher matcher = pattern.matcher(fileName);
        return matcher.matches();
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new ImageUpload().setVisible(true);
            }
        });
    }
}
```

Resultados esperados:



6.6. Validación de contraseñas en Java

Ejemplo de una aplicación en Java para generar contraseñas aleatorias y validar si cumplen con ciertos requisitos de seguridad. El programa genera contraseñas y verifica si cumplen con una política de contraseñas segura, además de proporcionar una razón si no son válidas.

Se busca una cadena de caracteres que cumpla con los siguientes requisitos:

- Contener al menos una letra mayúscula.
- Contener al menos una letra minúscula.
- Contener al menos un número.
- Contener al menos un carácter especial.
- Tener una longitud mínima de 8 caracteres.

Expresiones usadas:

```
(?=.*[A-Z])              Al menos una letra mayuscula
(?=.*[a-z])              Al menos una letra minuscula
(?=.*\\d)                Al menos un digito
(?=.*[@$!%*?&])          Al menos un caracter especial
[A-Za-z\\d@$!%*?&]{8,}   Al menos 8 caracteres de letras,
                         numeros o caracteres especiales
```

Código Java:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Password {

    public static void main(String[] args) {
        new Password();
    }

    public Password() {
        for (int i = 0; i < 10; i++) {
            String password = generarPassword();
            System.out.println("Contrasenia generada:" + password)
    ;
```

```
            if (validarPassword(password)) {
                System.out.println("Contrasena valida\n\n");
            } else {
                System.out.println("Contrasena no valida\nRazon:");
                razon(password);
                System.out.println("\n");
            }
        }
    }

    boolean validarPassword(String password) {
        String patron = "^(?=.*[A-Z])(?=.*[a-z])(?=.*\\d)(?=.*[@$
    !%*?&])[A-Za-z\\d@$!%*?&]{8,}$";
        Pattern pattern = Pattern.compile(patron);
        Matcher matcher = pattern.matcher(password);
        return matcher.matches();
    }

    void razon(String password) {
        String regexNum="(?=.*\\d)";
        String regexMayus="(?=.*[A-Z])";
        String regexMinus="(?=.*[a-z])";
        String regexCaracter="(?=.*[@$!%*?&])";
        String regexCantidad="[A-Za-z\\d@$!%*?&]{8,}";
        Pattern pattern = Pattern.compile(regexNum);
        Matcher matcher = pattern.matcher(password);
        if(!matcher.find()) {
            System.out.println("\tLa contrasena debe contener un
    numero");
        }
        pattern = Pattern.compile(regexMayus);
        matcher = pattern.matcher(password);
        if(!matcher.find()) {
            System.out.println("\tLa contrasena debe contener una
    mayuscula");
        }
        pattern = Pattern.compile(regexMinus);
        matcher = pattern.matcher(password);
        if(!matcher.find()) {
            System.out.println("\tLa contrasena debe contener una
    minuscula");
        }
        pattern = Pattern.compile(regexCaracter);
        matcher = pattern.matcher(password);
        if(!matcher.find()) {
            System.out.println("\tLa contrasena debe contener un
    caracter especial @$!%*?&");
        }
        pattern = Pattern.compile(regexCantidad);
        matcher = pattern.matcher(password);
        if(!matcher.find()) {
            System.out.println("\tLa contrasena debe tener al menos
    8 caracteres");
        }
    }

    String generarPassword() {
        String cadena = "
    abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890@$
    !%*?&";
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 7 +(int) (Math.random() * 10); i++) {
```

```
            int indice = (int) (Math.random() * cadena.length());
            sb.append(cadena.charAt(indice));
        }
        return sb.toString();
    }
}
```

Resultados esperados:

```
Contrasena generada: vBi@mtR%&sz1
Contrasena valida


Contrasena generada: @T2BAbJQ5lu
Contrasena valida


Contrasena generada: Uzq2pUw8tR8o
Contrasena no valida
Razon:
        La Contrasena debe contener un caracter especial @$!%*?&


Contrasena generada: B$tK@YNd
Contrasena no valida
Razon:
        La Contrasena debe contener un numero


Contrasena generada: fl&7Apf2
Contrasena valida


Contrasena generada: G&5FEfS12zi
Contrasena valida


Contrasena generada: j32D?3?rxM
Contrasena valida


Contrasena generada: MRKGOSTct
Contrasena no valida
Razon:
        La Contrasena debe contener un numero
        La Contrasena debe contener un caracter especial @$!%*?&


Contrasena generada: IlhKXP%&
Contrasena no valida
Razon:
        La Contrasena debe contener un numero


Contrasena generada: gEux0qf@61G
Contrasena valida
```

6.7. Web Scraping con Java

Ejemplo de una aplicación en Java para realizar web scraping utilizando expresiones regulares. La aplicación obtiene información de un estudiante a partir de su número de registro desde una página web y muestra el nombre, número de carnet y registro del estudiante.

Las expresiones regulares se utilizan para buscar patrones específicos en el contenido HTML de la página web. El programa realiza una solicitud HTTP para obtener el contenido de la página, luego aplica las expresiones regulares para extraer los datos deseados. Expresiones usadas:

```
Se usa as etiquetas <span> en html para mostar Nombre, CI, y
    regitro.
con sus respectivos id: lNombre, lci, lreg
Con expresiones regulare buscamos todo lo estaticomo como literal:
id="lNombre" style="display:inline-block;">

El contenido de dentro de la etiqueta:
(.*)

Y el cierre de la etiqueta:
</span>
```

Código Java:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Scrapping {
    public static void main(String[] args) {
        new Scrapping();
    }
    Scrapping(){
        System.out.println("Ejemplo practico de web scraping con
    regex");
        System.out.println("Obtiene datos de un estudiante usando
    el registro");
        String registro = "222106867";
        String url = "https://caja.uagrm.edu.bo/listado.aspx?idper=
    "+registro+"&tipoper=1&sem=&anio=";
        String html = fetch(url); //obtiene el html
        scrapp(html);
    }

    public void scrapp(String html){
        //regex para el nombre, carnet y Registro en el html
        Pattern regexNombre = Pattern.compile("id=\"lNombre\" style
    =\"display:inline-block;\">(.*?)</span>");
        Pattern regexCi = Pattern.compile("id=\"lci\" style=\"
    display:inline-block;\">(.*?)</span>");
        Pattern regexRegistro = Pattern.compile("id=\"lreg\" style
    =\"display:inline-block;\">(.*?)</span>");
        Matcher matcher;
        matcher = regexNombre.matcher(html);
```

```
        if(matcher.find())
            System.out.println("Nombre:"+matcher.group(1));
        matcher = regexCi.matcher(html);
        if(matcher.find())
            System.out.println("CI:"+matcher.group(1).replaceAll("
    -", ""));
        matcher = regexRegistro.matcher(html);
        if(matcher.find())
            System.out.println("Registro:"+matcher.group(1));
    }

    public static String fetch(String urlToRead){
        StringBuilder result = new StringBuilder();
        try {
            URL url = new URL(urlToRead);
            HttpURLConnection conn = (HttpURLConnection) url.
    openConnection();
            conn.setRequestMethod("GET");
            try (BufferedReader rd = new BufferedReader(new
    InputStreamReader(conn.getInputStream()))) {
                String line;
                while ((line = rd.readLine()) != null) {
                    result.append(line);
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return result.toString();
    }
}
```

Resultados esperados:

```
Ejemplo practico de web scraping con regex
Obtiene datos de un estudiante usando el registro
Nombre: POZO SOLIZ MARTHEL PEDRO
CI: 9057219-SCZ
Registro: 222106867
```

7. Conclusión

Las expresiones regulares son una gran herramienta en el ambito de la programacion, nos facilitan de muchas maneras el procesamiento de texto y usadas con creatividad nos pueden sacar de muchos problemas. Sin las expresones regulares nuestro codigo podria ganar mucha extension lo cual no restará legibilidad y limpieza.

8. Bibliografía

IBM, (2024).Expresiones Regulares. https://www.ibm.com/docs/en/i/7.5?topic=expressions-regular

Fonseca, Ivan. Aplicaciones de Expresiones Regulares, https://es.scribd.com/document/246172436/Aplica de-Expresiones-Regulares

Gskinner. Regexr, https://regexr.com/

Oracle Docs. Regular Expressions, https://docs.oracle.com/javase/tutorial/essential/regex/