**Trabajo práctico nro. 12**

|  |  |  |
| --- | --- | --- |
| Logo Departamento 2.jpg | **Asignatura: Laboratorio I** | |
|  | |
| **Cursado:**Primer Trimestre | **Horas** **semanales**: |
|  | **Horas semestrales:**  *Cantidad estimada de horas semestrales/anuales.* |
| **Carrera**: *Tecnicatura Universitaria en Programación* | **Nivel (Año):** |
| **Ciclo Lectivo: 2023** |

1. **Tema:**

**Expresiones Regulares**

1. **Enunciados:**

**Empezaremos por unos ejercicios básicos de programas Java con estructura secuencial, es decir, en estos programas no hay instrucciones condicionales ni repetitivas. En la mayoría de ellos las operaciones a realizar son: lectura de datos por teclado, realizar alguna operación con esos datos y mostrar resultados por pantalla.**

**Recomendaciones: Leer la teoría, investigar, consultar si existen dudas y realizar el trabajo en clases. La entrega de este trabajo practico es individual.**

1. **Investigación de Expresiones Regulares**:
   * Investiga y comprende los conceptos básicos de las expresiones regulares, incluyendo caracteres especiales, cuantificadores y grupos.
   * Explora ejemplos de patrones comunes de expresiones regulares utilizados en la validación de datos.

Las expresiones regulares son patrones de búsqueda utilizados para identificar y manipular cadenas de texto. Estos patrones son utilizados en diversas áreas, como procesamiento de texto, análisis de datos, búsqueda y sustitución de texto, validación de cadenas, entre otros.

Caracteres Especiales:

. (Punto): Representa cualquier carácter, excepto un salto de línea.

\ (Barra invertida): Se utiliza para escapar caracteres especiales o para denotar secuencias especiales (por ejemplo, \d para dígitos).

[ ] (Corchetes): Se utilizan para definir un conjunto de caracteres. Por ejemplo, [abc] coincidirá con 'a', 'b' o 'c'.

^ (Caret): Indica el inicio de una cadena. Por ejemplo, ^Hola coincidirá con "Hola mundo", pero no con "Un Hola".

$ (Dólar): Indica el final de una cadena. Por ejemplo, mundo$ coincidirá con "Hola mundo", pero no con "mundo, Hola".

| (Barra vertical): Se utiliza para indicar alternativas. Por ejemplo, gato|perro coincidirá con "gato" o "perro".

Cuantificadores:

+ (Uno o más): Coincide con una o más repeticiones del elemento anterior. Por ejemplo, a+ coincidirá con "a", "aa", "aaa", etc.

\* (Cero o más): Coincide con cero o más repeticiones del elemento anterior. Por ejemplo, a\* coincidirá con "", "a", "aa", "aaa", etc.

? (Cero o una): Indica que el elemento anterior es opcional. Por ejemplo, colou?r coincidirá con "color" o "colour".

{n} (Exactamente n): Coincide con exactamente n repeticiones del elemento anterior. Por ejemplo, \d{3} coincidirá con tres dígitos.

{n,} (Al menos n): Coincide con al menos n repeticiones del elemento anterior.

{n,m} (Entre n y m): Coincide con un rango específico de repeticiones del elemento anterior.

Grupos y Captura:

( ): Los paréntesis se utilizan para crear grupos. Pueden ser utilizados para aplicar cuantificadores a subpatrones específicos, o para capturar una porción del texto.

\n (Referencia de grupo): Después de una coincidencia exitosa, \n hace referencia al texto capturado por el grupo n.

1. **Creación de un Conjunto de Datos**:
   * Crea un conjunto de datos ficticio que contenga información variada, como direcciones de correo electrónico, números de teléfono, fechas, códigos postales, etc.

ArrayList<String> direccionesCorreo = new ArrayList<>();

ArrayList<String> numerosTelefono = new ArrayList<>();

ArrayList<String> fechas = new ArrayList<>();

ArrayList<String> codigosPostales = new ArrayList<>();

// Direcciones de Correo Electrónico

direccionesCorreo.add("paulog@hotmail.com");

direccionesCorreo.add("messi@gmail.com");

// Números de Teléfono

numerosTelefono.add("2616394540");

numerosTelefono.add("2616384234");

// Fechas

fechas.add("16/02/2000");

fechas.add("12/31/2023");

// Códigos Postales

codigosPostales.add("12345");

codigosPostales.add("54321");

1. **Validación de Datos**:
   * Diseña un programa en un lenguaje de programación (Java) que utilice expresiones regulares para validar y verificar los datos en tu conjunto de datos. Por ejemplo, puedes verificar si una dirección de correo electrónico es válida o si un número de teléfono sigue un formato específico.

String Correo = "^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}$";

Telefono = "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$";

// Patrones de las expresiones regulares

Pattern patronCorreo = Pattern.compile(Correo);

Pattern patronTelefono = Pattern.compile(Telefono);

// Validación de direcciones de correo electrónico

for (String email : direccionesCorreo) {

Matcher matcher = patronCorreo.matcher(email);

if (matcher.matches()) {

System.out.println("La dirección de correo electrónico " + email + " es válida.");

} else {

System.out.println("La dirección de correo electrónico " + email + " no es válida.");

}

}

// Validación de números de teléfono

for (String telefono : numerosTelefono) {

Matcher matcher = patronTelefono.matcher(telefono);

if (matcher.matches()) {

System.out.println("El número de teléfono " + telefono + " sigue un formato válido.");

} else {

System.out.println("El número de teléfono " + telefono + " no sigue un formato válido.");

}

}

* + Validar entrada de datos en un arraylist y aplicarlo.

1. **Informe de Investigación**:
   * Prepara un informe que desarrolle el tema sobre las expresiones regulares y cómo se aplicaron en la validación de datos.
   * Incluye ejemplos de patrones de expresiones regulares utilizados en tu programa y cómo ayudaron en la validación de los datos.
   * Comprensión de los conceptos de expresiones regulares.
   * Diseño y desarrollo del programa de validación de datos.
   * Claridad y organización del informe de investigación.

Las expresiones regulares son patrones de búsqueda utilizados para identificar y manipular cadenas de texto. Permiten definir reglas específicas para encontrar, filtrar o manipular cadenas de caracteres, brindando un poderoso mecanismo para validar y procesar datos de forma eficiente.

Conceptos de Expresiones Regulares

-Caracteres Literales: Representan caracteres que coinciden exactamente consigo mismos.

Caracteres Especiales: Símbolos con significados especiales, como el punto (.), el asterisco (\*), el paréntesis (), entre otros.

Cuantificadores: Especifican la cantidad de repeticiones de un carácter o grupo de caracteres.

-Grupos y Captura Permiten agrupar patrones y capturar subcadenas específicas.

Ejemplos de Patrones de Expresiones Regulares

Patrón de Correo Electrónico:

String Correo = "^[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}$";

Este patrón verifica si una cadena coincide con un formato típico de correo electrónico. Verifica la presencia de caracteres alfanuméricos seguidos de un '@', un dominio y un dominio de nivel superior.

Patrón de Número de Teléfono:

String Telefono = "^\\(?(\\d{3})\\)?[- ]?(\\d{3})[- ]?(\\d{4})$";

Este patrón valida números de teléfono en formatos comunes, permitiendo paréntesis opcionales para el código de área y guiones o espacios en posiciones específicas.

Diseño y Desarrollo del Programa de Validación

El programa de validación utiliza los patrones de expresiones regulares para validar datos ficticios de correos electrónicos y números de teléfono.

- Define los patrones con `Pattern.compile()`.

- Itera sobre los conjuntos de datos a validar.

- Utiliza `Matcher.matches()` para verificar si cada dato coincide con el patrón.

Cómo las Expresiones Regulares Ayudaron en la Validación

Las expresiones regulares proporcionan un método sistemático y flexible para definir patrones de validación. En el programa, permiten una validación eficiente de los datos de correo electrónico y números de teléfono.

Las expresiones regulares ofrecen la capacidad de definir reglas específicas que los datos deben seguir para ser considerados válidos. Proporcionan flexibilidad al definir patrones complejos para verificar múltiples formatos de datos.

Conclusión

Las expresiones regulares son una herramienta poderosa en la validación y manipulación de datos. En el contexto del programa desarrollado, permitieron verificar de manera efectiva la validez de direcciones de correo electrónico y números de teléfono, aplicando patrones predefinidos.

La comprensión de los conceptos de expresiones regulares y su aplicación práctica en la validación de datos demuestra su utilidad en entornos de desarrollo de software y manipulación de información.

1. **Ejercicios**:

* Comprobar si el String *cadena* no acaba con un dígito.
* Comprobar si el String *cadena* no empieza por un dígito.
* Comprobar si el String *cadena* está formado por un mínimo de 5 letras mayúsculas o minúsculas y un máximo de 10.
* Validar el ingreso de un DNI (xx-xxx-xxx).
* Guardar en un arraylist una serie de mails, pero antes de ser guardados deben cumplir con las condiciones de las expresiones regulares.
* Comprobar si el String *cadena* contiene “abc”.

https://github.com/paulog16/LabComputacion