INTRODUCCION A LA TEORÍA DE LA COMPUTACIÓN

PROYECTO DE I-CORTE

Brayan Andres Garzon Lopez

Carlos Andres Cruz Casas

Sara Sofia Lis Moreno

Universidad Central
2021
Julio César Sierra G.

Contenido

1. Implementación de ER

Para el siguiente proyecto se realizo la implementación de las siguientes Expresiones Regulares:

Expresión	Descripción
.	En el modo predeterminado, esto coincide con cualquier carácter excepto con una nueva línea. Si se ha especificado el indicador DOTALL, esto coincide con cualquier carácter que incluya una nueva línea.
^	Coincide con el comienzo de la cadena, y en modo MULTILINE también coincide inmediatamente después de cada nueva línea.
$	Coincide con el final de la cadena o justo antes de la nueva línea al final de la cadena, y en modo MULTILINE también coincide antes de una nueva línea. foo coincide con “foo” y “foobar”, mientras que la expresión regular foo$ sólo coincide con “foo”. Más interesante aún, al buscar foo.$ en 'foo1\nfoo2\n' coincide con “foo2” normalmente, pero solo “foo1” en MULTILINE`; si busca un solo $ en 'foo\n' encontrará dos coincidencias (vacías): una justo antes de una nueva línea, y otra al final de la cadena.
*	Hace que el RE resultante coincida con 0 o más repeticiones del RE precedente, tantas repeticiones como sean posibles. ab* coincidirá con “a”, “ab” o “a” seguido de cualquier número de “b”.
+	Hace que la RE resultante coincida con 1 o más repeticiones de la RE precedente. ab+ coincidirá con “a” seguido de cualquier número distinto de cero de “b”; no coincidirá solo con “a”.
?	Hace que la RE resultante coincida con 0 o 1 repeticiones de la RE precedente. ab? coincidirá con “a” o “ab”.
\	O bien se escapan a los caracteres especiales (lo que le permite hacer coincidir caracteres como '*', '?', y así sucesivamente), o se señala una secuencia especial; las secuencias especiales se explican más adelante.
[]	Se utiliza para indicar un conjunto de caracteres.

|	A|B, donde A y B pueden ser RE arbitrarias, crea una expresión regular que coincidirá con A or B. Un número arbitrario de RE puede ser separado por '|' de esta manera. Esto puede también ser usado dentro de grupos (ver más adelante). Cuando la cadena de destino es procesada, los RE separados por '|' son probados de izquierda a derecha. Cuando un patrón coincide completamente, esa rama es aceptada. Esto significa que una vez que A coincida, B no se comprobará más, incluso si se produce una coincidencia general más larga. En otras palabras, el operador de '|' nunca es codicioso. Para emparejar un literal '|', se usa \\\|, o se envuelve dentro de una clase de caracteres, como en [|].
(...)	Coincide con cualquier expresión regular que esté dentro de los paréntesis, e indica el comienzo y el final de un grupo; el contenido de un grupo puede ser recuperado después de que se haya realizado una coincidencia, y puede coincidir más adelante en la cadena con la secuencia especial \number, que se describe más adelante. Para hacer coincidir los literales `'(' o ')', se usa \(o \), o se envuelve dentro de una clase de caracteres: [(], [)].

2. Diagramas UML

A continuación, agregamos los distintos diagramas UML para la comprensión del proyecto en cuestión:

Diagrama de clases:

Diagrama de estados

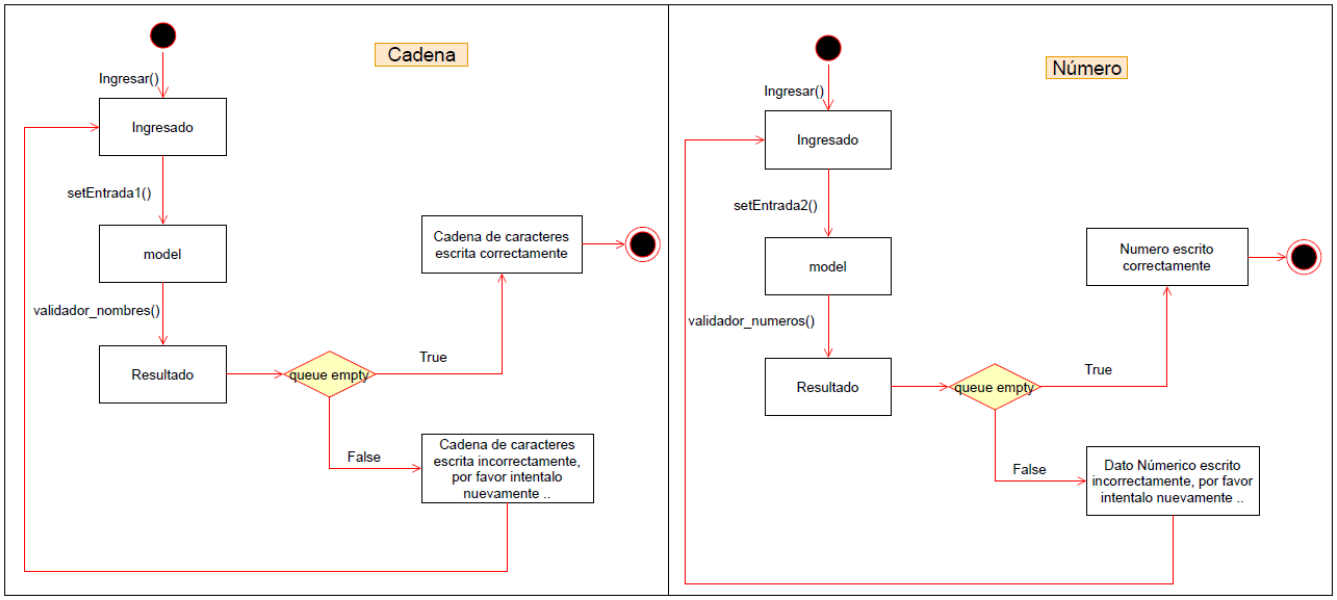
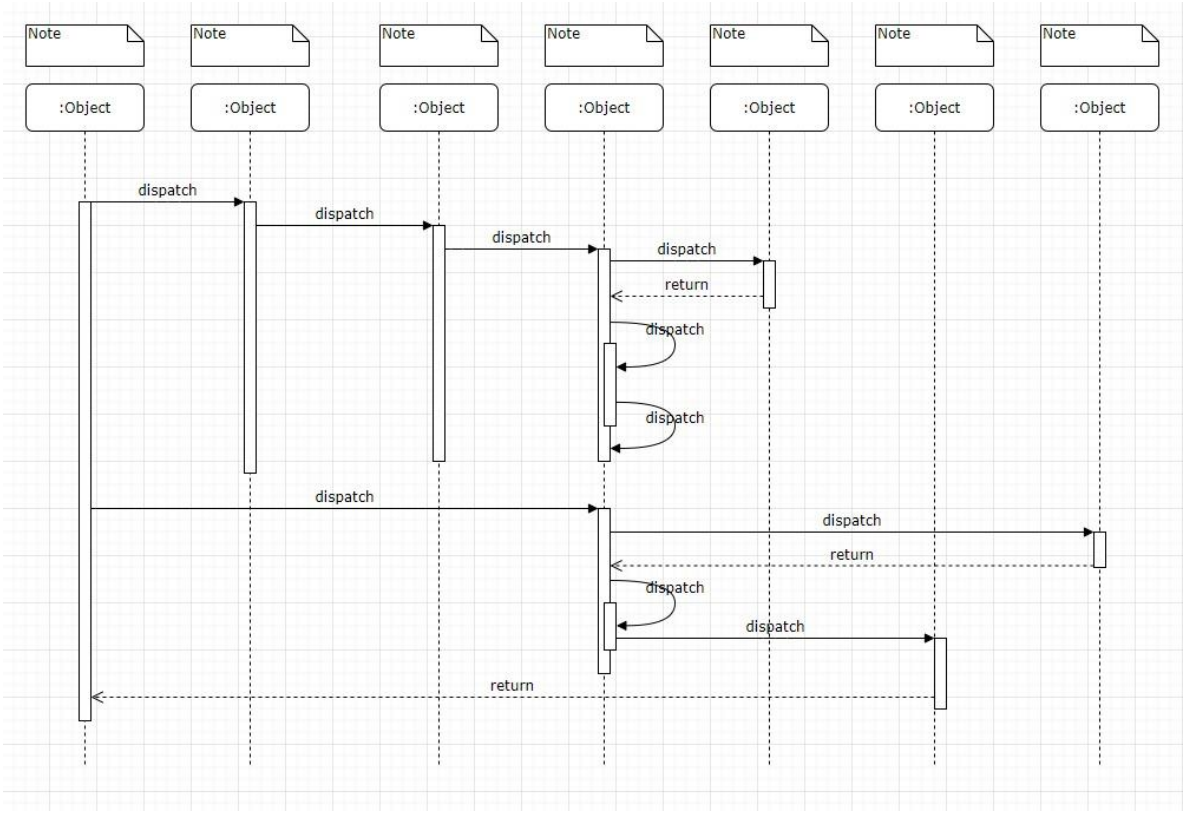


Diagrama de secuencia:



3. Código fuente:

A continuación, encontrara el código fuente del proyecto realizado:

View.py

```
from tkinter import *
import model as m
import  controller as c

model = m.Ingreso()
controller = c.Validador()

root = Tk()
root.title('PROYECTO DE I-CORTE')
root.geometry("800x800")

# funcion1
def clic():
    Str()
    Int()
    Mail()
    Not()

def Str():
        ingresado = str(e.get())
        model.setEntrada1(ingresado)
        validador = controller.validador_nombres(model)
        if bool(validador) == True:
            answer.config(text="Cadena de caracteres escrita correctamente", fg='#256614')
        else:
            answer.config(text="Cadena de caracteres escrita incorrectamente, por favor intentalo nuevamente..", fg='#C33819')
```

```
def Int():
    ingresado = str(e2.get())
    model.setEntrada2(ingresado)
    validador = controller.validador_numeros(model)
    if bool(validador) == True:
        answer2.config(text="Numero escrito correctamente",fg='#256614')
    else:
        answer2.config(text="Dato Númerico escrito incorrectamente, por favor intentalo nuevamente..",fg='#C33819')



def Mail():
    ingresado = str(e3.get())
    model.setEntrada3(ingresado)
    validador = controller.validador_correos(model)
    if bool(validador) == True:
        answer3.config(text="Correo electronico escrito correctamente",fg='#256614')
    else:
        answer3.config(text="El correo no cumple con el formato correcto (usuario@dominio),por favor intentalo nuevamente..",fg='#C33819')


def Not():
    ingresado = str(e4.get())
    model.setEntrada4(ingresado)
    validador = controller.validador_notacion(model)
    if bool(validador) == True:
        answer4.config(text="Notación cientifica escrita correctamente",fg='#256614')
    else:
```

```
        answer4.config(text="Notación cientifica escrita incorrectamente, por favor intentalo nuevamente..",fg='#C33819')


#---------------------------------------------------------------------
# Titulo_Principal
my_label1 = Label(root, text="Validador de expresiones regulares",font=("Helvetica", 20))
my_label1.pack(pady=20)

#---------------------------------------------------------------------
# Titulo_1
my_label2 = Label(root, text="Cadena de Caracteres", font=("Helvetica", 10))
my_label2.pack(pady=20)
# campo1
e = Entry(root, width=40, font=("Arial", 10))
e.pack()
#response1
answer= Label(root, text="", font=("Helvetica", 10))
answer.pack(pady=20)

#---------------------------------------------------------------------
# Titulo_2
my_label3 = Label(root, text="Números",font=("Helvetica", 10))
my_label3.pack(pady=20)
# campo2
e2 = Entry(root, width=40, font=("Arial", 10))
e2.pack()
#response2
answer2= Label(root, text="", font=("Helvetica", 10))
answer2.pack(pady=20)
```

```
#-----------------------------------------------------------------
# Titulo_4
my_label5 = Label(root, text="Notación Cientifica", font=("Helvetica", 10))
my_label5.pack(pady=20)
# campo4
e4 = Entry(root, width=40, font=("Arial", 10),)
e4.pack()
#response4
answer4= Label(root, text="", font=("Helvetica", 10))
answer4.pack(pady=20)

#-----------------------------------------------------------------
# Titulo_3
my_label4 = Label(root, text="Correo Electronico", font=("Helvetica", 10))
my_label4.pack(pady=20)
# campo3
e3 = Entry(root, width=40, font=("Arial", 10),)
e3.pack()
#response3
answer3= Label(root, text="", font=("Helvetica", 10))
answer3.pack(pady=20)

#-----------------------------------------------------------------
# Boton
myButton = Button(root, text="Validar", command=clic)
myButton.pack(pady=20)
root.mainloop()
```

Model.py

```
#model

class Ingreso():


    __entrada1 = ""

    __entrada2 = ""

    __entrada3 = ""

    __entrada4 = ""

    __regex1 = "[a-zA-Z]"

    __regex2 = "^[+-]?([0-9]+([.][0-9]*)?|[.][0-9]+)$"

    __regex3 = "^[^@]+@[^@]+\.[a-zA-Z]{2,}$"

    __regex4 = "^[+-]?([0-9]*[.])?[0-9]+([eE][-+]?\d+)$"


    #Getters

    def getEntrada1(self):

        return self.__entrada1


    def getEntrada2(self):

        return self.__entrada2


    def getEntrada3(self):

        return self.__entrada3


    def getEntrada4(self):

        return self.__entrada4


    def getRegex1(self):

        return self.__regex1
```

```
    def getRegex2(self):
        return self.__regex2

    def getRegex3(self):
        return self.__regex3

    def getRegex4(self):
        return self.__regex4

    #Setters
    def setEntrada1(self, param):
        self.__entrada1 = param

    def setEntrada2(self, param):
        self.__entrada2 = param

    def setEntrada3(self, param):
        self.__entrada3 = param

    def setEntrada4(self, param):
        self.__entrada4 = param
```

Controller.py

```
#Controller
import re
from tkinter import *
class Validador:
```

```
def validador_nombres(self, Ingreso):
    regex0 = re.compile(Ingreso.getRegex1())
    print(f"regex: {regex0}")
    cadena0 = (Ingreso.getEntrada1())
    print(f"cadena: {cadena0}")
    Ingreso.setEntrada1("")
    resultado = regex0.match(cadena0)
    if bool(resultado) == True:
        return True
    else:
        return False

def validador_numeros(self, Ingreso):
    regex1 = re.compile(Ingreso.getRegex2())
    print(f"regex: {regex1}")
    cadena1 = Ingreso.getEntrada2()
    print(f"cadena: {cadena1}")
    Ingreso.setEntrada2("")
    resultado = regex1.match(cadena1)
    if bool(resultado) == True:
        return True
    else:
        return False

def validador_correos(self, Ingreso):
    regex2 = re.compile(Ingreso.getRegex3())
    print(f"regex: {regex2}")
```

```
    cadena2 = Ingreso.getEntrada3()
    print(f"cadena: {cadena2}")
    Ingreso.setEntrada3("")
    resultado = regex2.match(cadena2)
    if bool(resultado) == True:
        return True
    else:
        return False

def validador_notacion(self, Ingreso):
    regex4 = re.compile(Ingreso.getRegex4())
    print(f"regex: {regex4}")
    cadena4 = Ingreso.getEntrada4()
    print(f"cadena: {cadena4}")
    Ingreso.setEntrada4("")
    resultado = regex4.match(cadena4)
    if bool(resultado) == True:
        return True
    else:
        return False
```