Trabajo Práctico 0

3 de Marzo de 2016

Objetivos

1. Familiarizarse con el ambiente de desarrollo propuesto por la cátedra.
2. Resolver un problema sencillo. No vamos a pedir que el TP se resuelva correctamente desde el punto de vista de diseño.

Enunciado

Dada una regular expression y un entero n, se deben generar n strings que matcheen la regular expression dada. Las características que deben tener dichos strings se explican más abajo.

Nos enteramos de que no todos sabían que era una regular expression, por lo tanto vamos a explicar el tp sin explicar qué diablos es un regular expression, ya que en éste caso creemos que no aporta a la claridad del tp.

La regular expression de entrada es una secuencia de caracteres que incluye SOLAMENTE:

- **. (punto)**: Genera un caracter aleatoriamente. Si se limita a caracteres ASCII, esto es un caracter random entre el ASCII 0 y el ASCII 255. Ejemplo: “.” genera strings de la forma “a”, “b”, “h”, “4”, “!”, etc.
- **[<char$_0$><char$_1$>...<char$_n$>] (conjunto)**: Genera un caracter aleatoriamente, pero restringido a los caracteres especificados entre corchetes. Ejemplo: “[abc]” genera strings de la forma “a”, “b”, “c”, y solo esos.
- **Literales**: Genera el caracter especificado y solo ese caracter. Si es un caracter reservado se puede escapar anteponiendo backslash. Ejemplo: “a” produce un solo string “a”, “\[“ produce un solo string “[“.
- **? (Cuantificador cero o uno)**: Genera de forma aleatoria cero o una ocurrencia del caracter que lo precede. Ejemplo: “a?” Produce solo dos strings diferentes “” y “a”.
- *** (Cuantificador cero o muchos)**: Genera de forma aleatoria cero o muchas ocurrencias del caracter que lo precede. En éste caso debemos imponer algún límite a cuantos caracteres se generan. Éste valor se pasa como parámetro en el constructor de la clase que genera los strings. Ejemplos, “a*” genera strings de la forma “”, “a”, “aa”, “aaa”, “aaaa”, etc.

- **+ (Cuantificador uno o muchos)**: Genera de forma aleatoria una o muchas ocurrencias del caracter que lo precede. En éste caso, como en el anterior, debemos imponer algún límite a cuantos caracteres se generan. Ejemplo, “a+” genera strings de la forma “a”, “aa”, “aaa”, “aaaa”, etc.

Un ejemplo más complejo

Cuando describamos el procesamiento de la regular expression vamos a hablar de tokens en vez de caracteres, ya que un token puede estar formado por varios caracteres. La regular expression **“..+[ab]*d?c”** se procesaría de la siguiente manera:

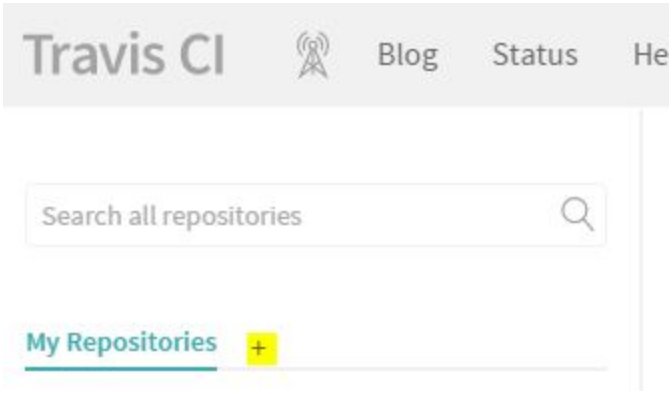
1. El primer token es un punto, por lo cual se genera cualquier caracter de forma aleatoria, digamos “f”.
2. El segundo token es un punto, pero con un cuantificador uno o muchos, entonces se genera aleatoriamente uno o muchos caracteres aleatorios. Tenemos que generar un número aleatorio entre 1 y N (el límite del “muchos” explicado arriba), supongamos que nos devuelve 5. Entonces generamos 5 caracteres aleatorios. Nuestro string nos queda “f**a!0nT**”.
3. El tercer token es un conjunto [ab] seguido por un cuantificador cero o muchos. Ahora tenemos que generar de forma aleatoria cero o muchas instancias de los caracteres del conjunto. Generamos un numero aleatorio entre cero y N, supongamos que nos devuelve 3. Entonces generamos 3 caracteres tomados del conjunto pero de forma aleatoria: ‘a’, ‘b’. ‘b’. Nuestro string nos queda “fa!0nT**abb**”
4. El cuarto token es un literal (d) seguido por un cuantificador cero o uno,. Entonces tenemos que generar de forma aleatoria cero o una instancia del literal. Supongamos que nos devuelve cero, por lo cual no generamos ningun caracter y nuestro string sigue siendo “fa!0nTabb”.
5. El quinto token es un literal sin cuantificador, por lo que simplemente se genera el literal. Nuestro string final nos queda “fa!0nTabb**c**”.
6. Éste proceso se repite n veces, para generar los n strings de salida.

Entrega

La entrega se realizará el lunes **21 de Marzo**, simplemente creando un tag llamado **Entrega**.

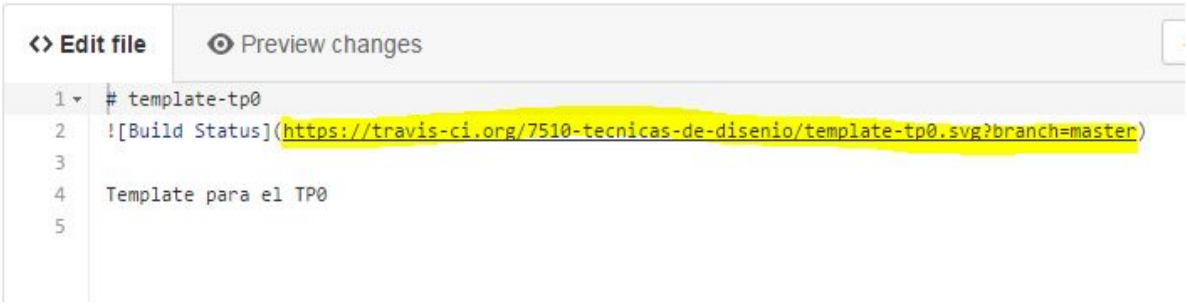
Algunas instrucciones

1. Hacer un fork (NO un clone) del repositorio https://github.com/7510-tecnicas-de-disenio/template-tp0. Luego de realizado el fork deberían poder ver el repositorio en sus cuentas de GitHub.
2. Importar el proyecto a IntelliJ como Gradle Project.
3. Deberían poder hacer ./gradlew build y el proyecto debería compilar.
4. En el código hay varias partes comentadas con un TODO. Dichas partes se deberían descomentar a medida que se avanza con el TP. El problema es que si no se comentaban la compilación tira warnings.
5. Hay un proyecto de Tests con algunos test comentados (están comentados porque como no hay implementación, fallan). Los pueden ir des-comentando a medida que se avanza con el desarrollo. También es OBLIGATORIO agregar más tests.
6. Servidor de Integración Continua. Durante el desarrollo de los TPs vamos a usar Travis (https://travis-ci.org/) como servidor de integración continua, principalmente porque se integra muy facilmente con GitHub. Van a ver que en proyecto hay un archivo que se llama .travis.yml. Este archivo tiene la configuración para poder compilar la aplicación el Travis. Deberían poder entrar a Travis-CI con su cuenta de GitHub y agregar el repositorio del proyecto.



7. Con cada commit, push y merge request, Travis lanza un nuevo build. Si el build es exitoso deberían ver una indicación:

Si hacen click sobre ésta imagen aparece una ventana con una URL. Tiene que copiar esa URL y reemplazar la que está en el README.md.



8. Cualquier duda, escribirla en el channel **#consultas-tp0**.