# Universidad de los Andes

Maestría en Biología Computacional

Fundamentos de Programación para Ciencias Biológicas

Modulo 4 - Laboratorio 4 (M4-L4)

**Cupi2**

## Objetivo

Aprender a ejecutar programas en Java desde línea de comandos y aprender a leer y escribir archivos.

## Pasos

**1. Crear un script en Java utilizando Eclipse.** No todos los programas que se escriben en Java tienen que tener interfaz gráfica o ser orientados por objetos. El objetivo del primer paso de esta guía es poder crear un programa sencillo en Java para, por ejemplo, leer el contenido de un archivo. Para esto se debe crear un proyecto en Eclipse, crear una clase con el script y ejecutar el script.

Para crear un proyecto en Eclipse se debe ir a File → new → Java Project. Esto despliega un diálogo para crear un proyecto nuevo. Solo deben escribir un nombre para el proyecto, por ejemplo “ScriptsLineaComandos” y dar clic en “Finish”. Esto crea una carpeta con el mismo nombre del proyecto en la ubicación del workspace que definieron al iniciar Eclipse con dos subcarpetas: “src” para los archivos .java y “bin” para los archivos “.class”. Esta última carpeta está oculta en la visualización de Eclipse.

El siguiente paso es crear la clase con el script. Similar a la guía de M4-L1 y al ejercicio del módulo 3, hacer clic derecho sobre la carpeta src y seleccionar New → Class. El nombre de la clase puede ser “EjemploScript1”. Hacer clic en la opción “public static void main (String [] args)” para crear un main automáticamente. El main es el primer método que se llama cuando se ejecuta un script, o en general cuando se ejecuta cualquier programa en Java.

Para ver cómo funciona con un ejemplo sencillo comencemos por el programa típico que imprime un mensaje en la pantalla. Para esto, escribir dentro del main el mensaje que quieran ver, por ejemplo:

```
System.out.println("Aprendí a programar este semestre");
```

Luego guardar el script y ejecutarlo utilizando el botón verde. Esto ejecuta el programa dentro del entorno de Eclipse y escribe el mensaje en la consola. Si no ven la consola, la pueden ver con la opción del menú Window → Show view → Console

Este script también se puede ejecutar desde la línea de comandos. Para esto se debe abrir una consola (en Windows también funciona) e ir a la carpeta del proyecto utilizando el comando “cd”. Luego de esto, escribir:

```
java -cp bin EjemploScript1
```

En este comando “EjemploScript1” es el nombre de la clase que contiene el script y “bin” es la carpeta en la que está el archivo “EjemploScript1.class” creado por Eclipse. Este archivo es la versión compilada de la clase que creamos y que procesa la máquina virtual de Java. Esto funciona de la misma forma para cualquier programa que se escriba en una sola clase.

Una de las maneras más comunes de recibir la entrada de un programa por línea de comandos es enviar argumentos de entrada después del comando que ejecuta el programa. En Java, esta función la cumple el parámetro “args” del main. Este parámetro es un arreglo de cadenas en el que cada cadena es un argumento que envía el usuario. Para ver esto, vamos a mejorar el script para que calcule el cuadrado de un número. Agregar al script las siguientes líneas:

```
String param1 = args[0];
int numero = Integer.parseInt(param1);
int cuadrado = numero*numero;
System.out.println("El cuadrado de "+numero+" es "+cuadrado);
```

Al intentar ejecutarlo desde Eclipse va a arrojar este error:

```
Exception in thread "main" Java.lang.ArrayIndexOutOfBoundsException: 0
	at EjemploScript1.main(EjemploScript1.Java:6)
```

Como el programa se está ejecutando sin argumentos, al intentar asignar el argumento “args[0]” a la variable “param1” falla porque el arreglo “args” está vacío. Lo mismo va a pasar si se ejecuta desde la línea de comandos utilizando nuevamente el comando escrito arriba. Para correr el script desde Eclipse se debe hacer clic en la flecha al lado del símbolo verde y seleccionar “Run configurations”. Esto despliega un diálogo en el que a la izquierda se debe seleccionar la clase con el script y a la derecha, en la pestaña “Arguments”, se debe escribir el número al que se le quiere calcular el cuadrado en la caja de texto “Program arguments”. Una vez escrito el número hacer clic en “Apply” y luego en “run”.

Desde la línea de comandos es más fácil escribir el argumento. Por ejemplo, para averiguar el cuadrado de 5 simplemente se debe ejecutar el programa así:

```
java -cp bin EjemploScript1 5
```

En la segunda línea, “param1” se convierte a entero para poder elevarlo al cuadrado. Si en lugar de un entero el programa recibe una cadena (por ejemplo “hola”), arroja este error:

```
Exception in thread "main" Java.lang.NumberFormatException: For input string:
"hola"
	at
Java.lang.NumberFormatException.forInputString(NumberFormatException.Java:65)
	at Java.lang.Integer.parseInt(Integer.Java:580)
	at Java.lang.Integer.parseInt(Integer.Java:615)
	at EjemploScript1.main(EjemploScript1.Java:7)
```

**2. Lectura de archivos de entrada.** En esta parte vamos a escribir un script para procesar un archivo de alineamientos de blast de la misma forma que lo hicimos con awk en el taller del módulo 2. La primera actividad de ese taller fue producir un reporte con el número de línea, el número de campos y la segunda columna, que es el id de la secuencia sobre la que alineó la consulta. Para comenzar, siguiendo los pasos del punto 1, crear una nueva clase llamada por ejemplo: “EjemploScriptArchivos” y agregar las siguientes líneas dentro del main:

```
//Ruta al archivo de entrada
String rutaArchivoEntrada = args[0];

try (FileReader fr = new FileReader(rutaArchivoEntrada);
     BufferedReader in = new BufferedReader(fr)) {
      //Lee la primera linea
      String linea=in.readLine();
      while(linea!=null) {
            System.out.println("Siguiente línea: "+linea);
            //Lee la siguiente linea
            linea=in.readLine();
      }
}
```

Este script simplemente imprime el contenido del archivo en la pantalla (como el comando “cat” de Linux). Sin embargo, tiene todo lo necesario para poder procesar correctamente cualquier archivo de texto. El script lee el archivo línea por línea en la variable “linea”, hasta que esta variable sea nula, con lo cual se indica que se llegó al final del archivo. Dentro del ciclo se realiza el procesamiento que sea necesario para la línea actual. Esta versión utiliza además una característica de Java 1.7 que se llama “try with resources” y que automáticamente cierra el canal de comunicación con el archivo una vez este ha sido procesado. Para verlo funcionando, descarguen el archivo de ejemplo “test.blast” en la carpeta del proyecto y ejecuten por línea de comandos:

```
java -cp bin EjemploScriptArchivos test.blast
```

Si el archivo que quieren procesar no está en la carpeta del proyecto, el parámetro puede tener la ruta completa. Sin embargo, si la ruta tiene espacios en blanco, se deben usar comillas para evitar que el espacio en blanco se interprete como otro parámetro diferente. Si el programa no puede encontrar el archivo genera el siguiente error:

```
Exception in thread "main" Java.io.FileNotFoundException: archivoQueNoExiste.txt
(No such file or directory)
      at Java.io.FileInputStream.open0(Native Method)
      at Java.io.FileInputStream.open(FileInputStream.Java:195)
      at Java.io.FileInputStream.<init>(FileInputStream.Java:138)
      at Java.io.FileInputStream.<init>(FileInputStream.Java:93)
      at EjemploScriptArchivos.main(EjemploScriptArchivos.Java:14)
```

Para generar el reporte que generábamos con awk, el paso más importante es utilizar el método “split” de la clase String para partir la línea en tokens o palabras que en la salida de blast representan las columnas. El número de línea lo podemos calcular simplemente usando

un contador igual a los que usamos en el módulo anterior. Reemplace el ciclo anterior (while) con el siguiente ciclo:

```
String linea=in.readLine();
for(int i=0;linea!=null;i++) {
      //Parte la linea en tokens, palabras, o columnas
      String [] columnas = linea.split("\t");
      //Escribe la informacion del reporte
      System.out.println(""+(i+1)+"\t"+columnas.length+"\t"+columnas[1]);
      //Leer la siguiente linea
      linea=in.readLine();
}
```

Al ejecutar de nuevo el programa debería salir un reporte igual al primer reporte que hicimos con awk. La siguiente tabla resume la correspondencia de variables entre awk y este script.

| Variable del script | Variable de awk | Descripción |
|---|---|---|
| linea | $0 | Línea completa |
| columnas | $1,$2,…,$NF | Cada uno de los tokens, palabras o columnas del archivo |
| i+1 | NR | Número de línea del archivo |
| columnas.length | NF | Número de tokens, palabras o columnas del archivo |

Es importante notar que mientras los índices en awk comienzan en 1, los índices en Java comienzan en cero; por esto, la primera columna en awk es $1 pero la primera columna en el script en Java es columnas[0]. Noten también que en Java se imprime (i+1) y no i por lo que el contador en Java se está iniciando en cero.

Mas allá de esto, como tanto la línea completa, como cada columna son cadenas, se pueden utilizar todos los métodos de la clase String y en general todo lo aprendido en el curso (condicionales, ciclos, métodos, etc) para procesar la información. Como ejercicios adicionales pueden intentar reproducir en Java los problemas incluidos en el taller de awk. Para esto es útil recordar que una cadena se interpreta como entero utilizando la función “Integer.parseInt” y como número real utilizando “Double.parseDouble”. Por ejemplo, para cargar correctamente la longitud del alineamiento y el porcentaje de identidad, dentro del ciclo se escribe:

```
int longitudAln = Integer.parseInt(columnas[3]);
double pctAln = Double.parseDouble(columnas[4]);
```

y ya con esto se pueden operar estas variables (sacar sumas y promedios, filtrar, etc) utilizando todo lo visto en el curso.

**3. Escritura de archivos**. Como aprendimos en el taller de awk, si un script o programa en Java se ejecuta en línea de comandos, la salida se puede redirigir a un archivo utilizando el operador “>” seguido del nombre del archivo de salida (funciona también en Windows):

```
java -cp bin EjemploScriptArchivos test.blast > test.out
```

Sin embargo, en muchos casos es necesario que el programa como tal genere el archivo de salida, por ejemplo porque queremos generar más de un archivo. Para esto, la clase de Java que permite generar archivos de texto más fácilmente es “PrintStream”. Lo primero que hay que hacer es agregarle al script un parámetro para el archivo de salida, declarar una variable para el flujo de salida y crear el objeto de la clase PrintStream dentro del try. Las líneas antes del ciclo quedarían así:

```
//Ruta al archivo de entrada
String rutaArchivoEntrada = args[0];
//Ruta al archivo de salida
String rutaArchivoSalida = args[1];
try (FileReader fr = new FileReader(rutaArchivoEntrada);
     BufferedReader in = new BufferedReader(fr);
     PrintStream out = new PrintStream(rutaArchivoSalida)) {
     //Lee la primera linea
     String linea=in.readLine();
```

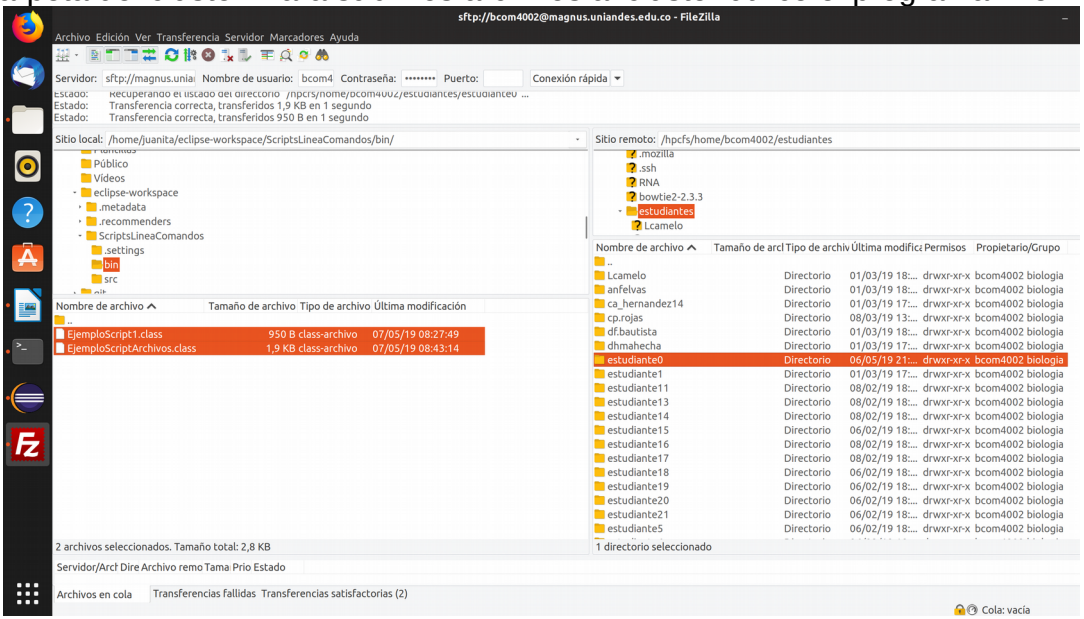
Finalmente, solo se debe quitar “System.” de la línea dentro del ciclo en la que se imprimía el reporte en pantalla para que imprima al archivo de salida. La línea quedaría así:

```
out.println(""+(i+1)+"\t"+columnas.length+"\t"+columnas[1]);
```

Noten que lo que se hace ahora es usar el objeto “out” de la clase “PrintStream” en lugar de “System.out” para imprimir. “System.out” es de por si un objeto de la clase “PrintStream” que representa un flujo de salida especial que imprime a la salida estándar (o sea a la pantalla). Para ejecutar el script se debe ahora proveer una ruta al archivo de salida:

```
java -cp bin EjemploScriptArchivos test.blast test.out
```

**4. Ejecución en el cluster.** Una de las premisas con las que se diseñó Java es que el código se escriba una sola vez y se ejecute en cualquier máquina con cualquier sistema operativo. Esto es así para cualquier sistema operativo en el que la máquina virtual de Java esté correctamente instalada, incluido el cluster. Para poder ejecutar los scripts de hoy en el cluster se deben subir los archivos EjemploScript1.class y EjemploScriptArchivos.class a alguna carpeta del cluster. Para subir los archivos al cluster utilice el programa FileZilla.



En el campo “servidor” escriba magnus.uniandes.edu.co, en “usuario” bcom4002, luego la contraseña y en “puerto” escriba 22. Seleccione los archivos correspondientes desde su computador (lado izquierdo) y cópielos a su carpeta de estudiante asignada en la carpeta del curso (lado derecho).

Luego, entrar al cluster y pedir un nodo interactivo. Desde Windows descargar el ejecutable de PuTTY (archivo putty.exe en http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) y escribir magnus en el campo llamado “hostname”. Desde MAC o Linux abrir una terminal y ejecutar:

```
ssh bcom4002@magnus.uniandes.edu.co
```

PuTTY solicitará el usuario y el password, desde MAC o Linux solo se solicitará el password.

Una vez se ingrese al nodo login del cluster la línea de comandos debería verse así:

```
[bcom4002@magnus ~]$
```

**2. Reservar un nodo interactivo con recursos específicos**. Escribir el comando:

```
[bcom4002@magnus ~]$ salloc --nodes=2 --ntasks-per-node=4 --mem=8G
--time=01:00:00
```

Este comando permite alocar recursos y luego hacer uso de ellos, y genera la siguiente salida:

```
salloc: Granted job allocation 78239
salloc: Waiting for resource configuration
salloc: Nodes node-[2,22] are ready for job

El mensaje indica que los recursos solicitados fueron concedidos.
Ahora debe conectarse al nodo indicado, el cual dispone de los
recursos solicitados, para trabajar de manera interactiva. Por
ejemplo:

[bcom4002@magnus ~]$ ssh node-22
```

La línea de comandos queda así:

```
[bcom4002@<NODO> ~]$
```

Donde NODO corresponde al nodo que ha sido reservado (cambia para cada persona).

Para poder ejecutar los scripts de Java es necesario cargar primero el módulo correspondiente. Escriba el siguiente comando:

```
[bcom4002@<NODO> ~]$ module load jdk/1.8.0_31
```

Ahora ejecuten el script con un comando como este:

```
[bcom4002@<NODO> ~]$ java -cp /path/to/scripts/ EjemploScript1 5
```

donde "`/path/to/scripts/`" es la ruta hacia la carpeta en la que están los archivos .class. Si esta ruta no es correcta, se genera un error como este:

```
Error: Could not find or load main class EjemploScript1
```

Para ejecutar el segundo script recuerden que el archivo test.blast está disponible en la carpeta "/hpcfs/home/bcom4002/tallerAWK/" o en sus carpetas, como enlace simbólico.

No olvidar cerrar la sesión tanto en el nodo interactivo como en magnus una vez terminen de ejecutar los scripts con el comando exit.