TOLJava: Guía Práctica para el programador

Por ***Humberto Andrés Carralero López (hcarralero@gmail.com)***

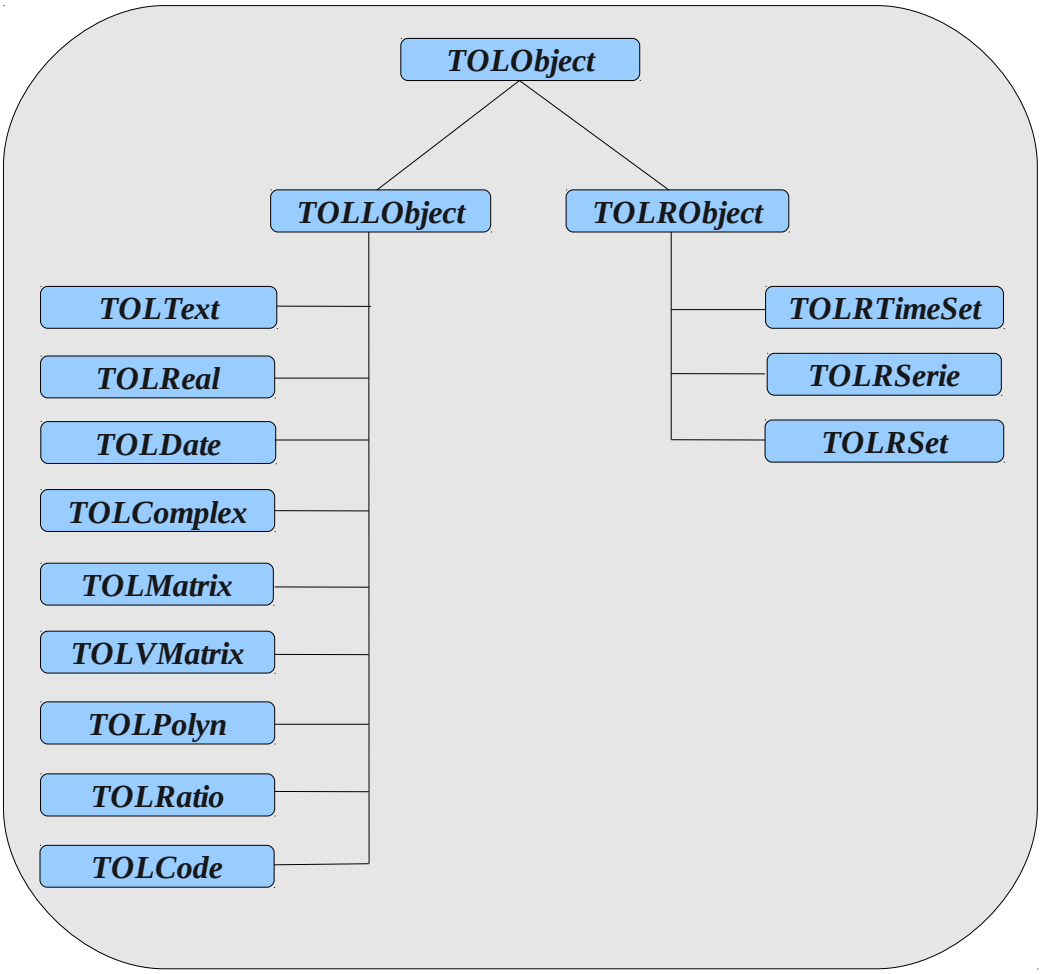
Tabla de Contenido

Introducción

Este breve guía te servirá de base para poder usar *TOLJava* en tus aplicaciones. Hemos resumido los pasos a seguir, los cuales están acompañados por ejemplos prácticos cuyo código fuente completo puedes descargar del sitio web demostrativo. Una explicación detallada de cada clase y exception con sus datos y métodos se encuentra en la *"Referencia del API de TOLJava"* que se incluye en la distribución de *TOLJava*.

Los fragmentos de código que se muestran en este manual han sido tomados de los 9 programas tipo consola que se incluyen como "*Programas Ejemplos*" en la distribución de *TOLJava*.

La biblioteca está integramente contenida en el paquete *com.hcarralero.toljava* y los elementos que la componen son ilustrados en el gráfico siguiente:



En el gráfico anterior observamos una jerarquía de clases que representan los tipos de datos existentes en *TOL*.

Inclusión de la Biblioteca *TOLJava*

Para poder hacer uso de *TOLJava* lo primero que se debe garantizar es que la biblioteca es accesible al programa, si usas un IDE como Netbeans o Eclipse, añade el archivo *TOLJava.jar* como una biblioteca al proyecto donde se vaya a utilizar; para otros entornos busca una opción similar o siempre tienes la opción de añadir este archivo al *CLASSPATH* de tu entorno de ejecución.

Una vez que la biblioteca es accesible podrás incluir cualquier clase que requieras en tu programa usando la sintaxis ***import com.hcarralero.toljava.<XXX>***, donde ***<XXX>*** es ******* para indicar todas las clases o el nombre específico de la clase que se requiera, por ejemplo, ***TOLMachine***.

Acceso al intérprete de *TOL*

Para que una aplicación tenga acceso a *TOL* debe crear un objeto de la clase ***TOLMachine***; con esto se cargará en memoria una instancia del intérprete de *TOL*, el código típico para hacer esto es:

```
TOLMachine tm= new TOLMachine();
```

Ahora se puede usar la instancia de ***TOLMachine* (*tm*)** para interactuar con el intérprete de *TOL*; la clase ***TOLMachine*** brinda una amplia variedad de métodos que permiten realizar todas las operaciones que requeriría una aplicación sobre un entorno *TOL*.

Ejecución de Código *TOL*

Para ejecutar código *TOL* se require usar el método ***execute***, el cual toma como parámetro una cadena de caracteres que contiene el código *TOL* que se quiere ejecutar. El siguiente fragmento de código muestra la codificación típica requerida para ejecutar código *TOL*:

```
//Se crea una instancia de la clase TOLMachine.
TOLMachine tm= new TOLMachine();
//En una cadena se almacena el código TOL que se ejecutará; en este caso
//el código es para definir las variables tipo VMatrix: ma, mb y mc.
 String code= "Text x=\"Hola. Bienvenido a TOLJava.\"; Real y= 3.5+7;";
Object res[]= tm.execute(code);
//Se invoca el intérprete de TOL para ejecutar el código pasado por parámetro.
//El resultado es un arreglo de cadenas.
System.out.println();
if (res==null) {
   //Si el resultado es nulo, entonces se ha producido un error en el acceso a TOL.
   System.out.println("ERROR ACCEDIENDO A TOL->execute");
} else {
   //Se ha logrado acceder al intérprete y ejecutar el código.
   int ne= 0;
   try {
      //La primera cadena representa el número de errores detectado por el Intérprete de TOL.
      ne= Integer.parseInt((String)res[0]);
      if (ne>0) {
         //El intérprete de TOL reportó al menos un error.
         System.out.println("Se ha(n) detectado "+ne+" error(es)");
      } else {
         //El intérprete de TOL no reportó errores.
```

```
            System.out.println("No se han detectado errores");
        }
        //Los siguientes objetos que conforman la respuesta del intérprete pueden ser:
        //Warnings, Errores o mensajes generados por el código TOL ejecutado.
        //Para procesar los mensajes generados debe comprender el formato de
        //estos mensajes. (Consulte la documentación de TOL).
        for (int i= 1; i<res.length; ++i) {
            //Cada objeto es una cadena
            String strRes= (String)res[i];
            //Se obtiene la longitud de la cadena.
            int l= strRes.length();
            if (strRes.charAt(0)=='<'&&strRes.charAt(2)=='>'&&(strRes.charAt(1)=='E'|| strRes.charAt(1)=='W')&&l>=7) {
                //En este caso el mensaje es un 'W'arning o un 'E'rror.
                System.out.println("\t"+strRes.substring(3, l-5));
            } else {
                //En este caso es un mensaje común.
                System.out.println("\t"+strRes);
            }
        }          } catch (NumberFormatException ex) {
        //Algo inusual (ERROR INTERNO DE TOLJava) ha ocurrido.
        System.out.println("ERROR DE ACCESO AL NÚMERO DE ERRORES");
    }
}
```

El bloque de código anterior tiene comentado cada una de las líneas por lo que no se requiere explicación adicional; sin embargo, sería bueno resaltar que en ese fragmento está el tratamiento general y se abordan todos los aspectos relacionados con la respuesta de *TOL*, la forma en que se dará tratamiento a cada uno de esos aspectos dependerá de los requerimientos específicos del programa donde se ejecute el código.

Acceso a Variables *TOL*

Desde *TOLJava* se puede acceder a cualquier objeto del intérprete de *TOL* que esté asignado a una variable. Desde el punto de vista de *TOLJava* existen dos categorías de objetos accesibles como variables definidas en *TOL*: objetos locales y referencias.

Un objeto local es un elemento que se crea dentro de la aplicación *Java* y cuyo valor puede ser tomado desde una variable definida en *TOL* o asignado en el mismo código *Java*; los objetos locales pueden ser instancias de las clases derivadas de ***TOLLObject***: ***TOLReal***, ***TOLText***, ***TOLDate***, ***TOLComplex***, ***TOLMatrix***, ***TOLVMatrix***, ***TOLPolyn*** y ***TOLRatio***. Los objetos locales se pueden crear de dos formas:

- Creando una instancia de la clase correspondiente con valores apropiados; por ejemplo, para crear un objeto local correspondiente al tipo *Date* de *TOL*, escribimos:

  ```
  //Se crea una instancia en Java correspondiente al tipo Date de TOL para la fecha "11 de Marzo de 2009"
  TOLDate idate= new TOLDate(2009, 3, 11);
  ```

- Accediendo a una instancia definida en el intérprete de *TOL*; por ejemplo, si en *TOL* se ha definido la variable *total* de tipo *Real*, para acceder a ella desde *Java* escribimos:

  ```
  //Se accede a la variable "total" definida en TOL
  TOLVariable vtotal= tm.getVariable("total");
  TOLReal total= (TOLReal)vtotal.getValue();
  ```

  En este caso, observe que se hace uso de la nueva clase ***TOLVariable*** que sirve para encapsular

cualquier variable definida en el intérprete de *TOL*; un objeto de esta clase almacena los atributos que hayan sido asignados a la variabel en *TOL*: nombre, descripción y valor.

La invocación del método ***getVariable*** podría lanzar una excepción en caso de que no exista ninguna variable con el nombre indicado o que la variable encontrada corresponda a un tipo de dato que no admita instancias locales como es el caso de los tipos de datos *Serie*, *TimeSet* y *Set*.

Independientemente de la forma en que se haya creado la variable, cambiar su valor como objeto *Java* no afecta a ninguna variable definida en el intérprete de *TOL*. Para cambiar el valor de una variable definida en *TOL* se requiere hacerlo de forma explícita usando el método ***setVariableValue***; por ejemplo, para cambiar el valor de la variable *"total"* definida en *TOL* escribimos:

```
//Se asigna a la variable "total" un nuevo valor
tm.setVariableValue("total", new TOLReal(4567.89));
```

La invocación del método ***setVariableValue*** podría lanzar una excepción en caso de que no se encontrara una variable con el nombre dado o el valor asignado sea de un tipo diferente al de la variable encontrada.

Manejo de variables tipo *Code*

Con *TOLJava* se puede realizar un manejo especial de las variables tipo *Code* que se hayan definido en el intéprete de *TOL*; a continuación mostramos a través de un ejemplo las operaciones que se pueden realizar con variables de tipo *Code*.

En primer lugar, supongamos que en *TOL* hemos definido las funciones *sumar* y *restar*, así como las variables *f* de tipo *Code* y *x* de tipo *Real*, el código *TOL* para realizar esto es:

```
Real sumar(Real a, Real b) {Real r= a+b; r};
Real restar(Real a, Real b) {Real r= a-b; r};
Code f= sumar;
Real x=5;
```

En el código anterior se le ha asignado a la variable *f* la función *sumar*, para asignarle otra función (digamos *restar*) usando *TOLJava* escribimos:

```
//Se asigna un nuevo valor a f, la función "restar" (el valor del campo información no se tiene en cuenta)
tm.setVariableValue( "f", new TOLCode("restar", null));
```

Después del código anterior la variable *f* tiene asignada la función *restar*, por lo que cualquier invocación a ella se traduciría en una llamada a la función *restar*.

En *TOLJava*, para invocar una función se requiere que esta sea asignada a una variable de tipo *Code* como se ha hecho en el ejemplo anterior; si la función asignada tiene parámetros como es el caso de *restar*, entonces antes de proceder a su invocación se requiere crear un arreglo de objetos con los parámetros de la invocación, cada elemento del arreglo puede ser un objeto local correspondiente al tipo de dato que espera la función o el nombre de una variable definida en *TOL* y del mismo tipo que el parámetro. El código siguiente muestra la invocación de la función *restar* por medio de la variable *f*, pasando como parámetros un valor de tipo *Real* y el nombre de una variable, también de tipo *Real*.

```
//Se crea un arreglo de objetos que servirá como lista de parámetros a la llamada a
//la función f, el 1er parámetro es una constante de Tipo TOLReal y el 2do es una
//cadena que representa el nombre de variable "x"
Object pars[]= {new TOLReal(12), "x"};
//Se hace una llamada a la función f pasando como argumentos el nombre de
//función "f" y el arreglo de parámetros. El resultado es asignado a la variable r3
```

```
TOLReal r3= (TOLReal)tm.callCode("f", pars);
//Se muestra el valor de r3 que es resultado de invocar la función f.
System.out.println("f(12, x)="+r3);
```

Manejo de variables de tipo "referencia"

Hay tres tipos de datos en *TOL* que no pueden ser tratados como objetos locales *Java*: *TimeSet, Serie* y *Set*; esto se debe a su complejidad dada por el tamaño, la forma en que se generan o el hecho de no ser finitos. Para estos tres tipos de datos en *TOLJava* se han definido clases que almacenan referencias a las variables correspondientes definidas en el intérprete de *TOL*; el manejo de cada uno de estos tipos de datos se expone en las siguientes tres secciones.

Al llamar cualquiera de los métodos que operan sobre estos tres tipos de datos se pueden lanzar excepciones relacionadas con la operación, tales como que el nombre de la variable no se encuentra, una fecha inválida, etc. En la "*Referencia del API de TOLJava*" se detallan todos los métodos y las condiciones bajo las cuales pueden ser lanzadas las excepciones.

Manejo de variables de tipo *TimeSet*

Una variable definida en *TOL* como *TimeSet* puede ser referenciada desde *Java* usando una instancia de la clase ***TOLRTimeSet***; en *TOLJava* se han creado las facilidades que han sido posibles y que son indispensables para operar un conjunto temporal desde una aplicación *Java*. El fragmento de código siguiente muestra la definición en *TOL* de un conjunto temporal con "todos los lunes del año 2011":

```
TimeSet ts1= WD(1)*Y(2011);
```

Tomando como base la definición del conjunto temporal *ts1* anterior, con *TOLJava* podemos realizar las siguientes operaciones:

- Obtener la primera fecha:

  ```
  //Para obtener la fecha inicial usaremos una fecha que sabemos es anterior a la primera fecha del TimeSet
  TOLDate f1= new TOLDate(2010, 12, 31);
  //Se obtiene la primera fecha del TimeSet buscando el sucesor de una fecha del año anterior
  TOLDate fini= tm.getTimeSetNextDate("ts1", f1);
  ```

  Observe que para usar esta técnica se requiere que el conjunto esté acotado inferiormente y conocer alguna fecha anterior a la primera fecha del conjunto temporal.
- Obtener la última fecha:

  ```
  //Para obtener la fecha final usaremos una fecha que sabemos es posterior a la última fecha del TimeSet
  TOLDate f2= new TOLDate(2012, 1, 1);
  //Se obtiene la última fecha del TimeSet buscando el antecesor de una fecha del año siguiente
  TOLDate ffin= tm.getTimeSetPrevDate("ts1", f2);
  ```

  Observe que para usar esta técnica se requiere que el conjunto esté acotado superiormente y conocer alguna fecha posterior a la primera fecha del conjunto temporal.
- Obtener la fecha sucesora o antecesora de una fecha dada:

  ```
  //Obtener el 2do lunes del año 2011 (fini es la primera fecha, o sea, el primer lunes del año 2011)
  TOLDate f2= tm.getTimeSetNextDate("ts1", fini);
  //Obtener el penúltimo lunes del año 2011 (ffin es la última fecha, o sea, el último lunes del año 2011)
  TOLDate fp= tm.getTimeSetPrevDate("ts1", ffin);
  ```

Usando estas técnicas se puede recorrer un conjunto temporal parcial o totalmente (si es acotado) para obtener sus elelementos.

Manejo de variables de tipo *Serie*

Una variable definida en *TOL* como *Serie* puede ser referenciada desde *Java* usando una instancia de la clase **TOLRSerie** y cada elemento de la serie es encapsulado en la clase ***TSItem***; en *TOLJava* se han creado las facilidades que han sido posibles y que son indispensables para operar una serie desde una aplicación *Java*. El fragmento de código siguiente muestra la definición en *TOL* de una serie aleatoria con valores para todos los días del año 2009:

```
Serie ser1 = SubSer( Gaussian( 0, 1 ), y2009m1d1, y2009m12d31 );
```

Tomando como base la definición de la serie *ser1* anterior, con *TOLJava* podemos realizar las siguientes operaciones:

- Obtener la fecha del primer valor de la serie:

```
//Obtención de la fecha del primer valor de la serie “ser1”
TOLDate fini= tm.getSerieFirstDate(“ser1”);
```

- Obtener la fecha del último valor de la serie:

```
//Obtención de la fecha del último valor de la serie “ser1”
TOLDate fini= tm.getSerieLastDate(“ser1”);
```

- Obtener un elemento de la serie en una fecha específica:

```
//Obtención del valor de la serie “ser1” en una fecha específica (“14 de Mayo de 2009”)
TOLTOLDate f= new TOLDate(2009, 5, 14);
double val= tm.getSerieValue(“ser1”, f);
```

- Obtener un arreglo de elementos de la serie a partir de una fecha dada:

```
//Se crea una variable fecha con valor dentro del rango de fechas de la serie (“21 de Abril de 2009”)
TOLDate date= new TOLDate(2009, 4, 21);
//Se obtiene un arreglo de a lo sumo 10 valores de la serie a partir de la fecha dada
TSItem tsis1[]= tm.getSerieItems("ser1", date, 10);
//Se imprime el número real de elementos obtenidos (pudiera ser inferior a 10)
System.out.println("Número de elementos leídos: "+tsis1.length);
```

Manejo de variables de tipo *Set*

Una variable definida en *TOL* como *Set* puede ser referenciada desde *Java* usando una instancia de la clase **TOLRSet**; en *TOLJava* se han creado las facilidades que han sido posibles y que son indispensables para operar una serie desde una aplicación *Java*. El fragmento de código siguiente muestra la definición en *TOL* de un conjunto que contiene 4 fechas:

```
Set  set1 = SetOfDate(y1995m12d3, y1996m11d15, y1994m1d4, y1994m12d04);
```

Tomando como base la definición del conjunto *set1* anterior, con *TOLJava* podemos realizar las siguientes operaciones:

- Obtener el tamaño del conjunto:

```
//Obtención del tamaño del conjunto “set1”
double size= tm.getSetSize(“se11”);
```

- Obtener un elemento del conjunto en una posición dada:

```
//Obtención del elemento ubicado en la posición 2 del conjunto “set1”
TOLObject ele2= tm.getSetElement(“set1”, 2);
```

- Obtener un arreglo con todos los elementos del conjunto:

```
//Obtención de todos los elementos del conjunto “set1”
TOLObject eles[]= tm.getSetElements(“set1”);
```

Las operaciones anteriores son suficientes para los requerimientos de cualquier aplicación que use un conjunto definido en TOL; debe tenerse en cuenta que cada elemento del conjunto puede ser de cualquiera de los tipos de datos permitidos en TOL, por lo que una vez que se obtenga uno o varios elementos de un conjunto se debe determinar el tipo de dato para procesarlo correctamente, una forma típica de hacer esto se ilustra en el fragmento de código siguiente:

```
//Obtención del elemento ubicado en la posición 2 del conjunto “set1”
TOLObject ele2= tm.getSetElement(“set1”, 2);
//Determinar si el valor obtenido es de tipo Real
if (ele2.getClass()==TOLReal.class) {
   //El elemento es de tipo Real por lo que podemos hacer un casting al tipo de dato exacto
   TOLReal r2= (TOLReal)ele2;
   …
}
```