Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Laboratorio de Algoritmos y Estructuras III
Trimestre Septiembre-Diciembre 2016

# Proyecto 1: Implementación del TAD Grafo

## 1. Descripción de la actividad

El objetivo de este proyecto es la familiarización con las operaciones básicas de los Tipos Abstractos de Datos (TADs) **Grafo**, **Grafo No Dirigido** y **Grafo Dirigido**. Para ello se desea que implemente los siguientes **TAD**s usando el lenguaje de programación `JAVA` y también que desarrolle una aplicación cliente que permita probar los **TAD**s. A continuación se describen los **TAD**s que se deben implementar.

### 1.1. El TAD *Vértice*

El **TAD** *Vértice* tiene en su representación un identificador de tipo `String` y un atributo de tipo `double` que es el peso asociado al vértice. Este TAD debe ser implementado como una clase concreta. Las operaciones mínimas que posee el **TAD** *Vértice* son las siguientes:

- `crearVértice`: (`String` $id$, `double` $p$) $\rightarrow$ `Vértice`
  Crea un nuevo vértice con un identificador $id$ y un peso $p$.
- `getPeso`: (`Vértice` $v$) $\rightarrow$ `double`
  Obtiene el peso del vértice $v$.
- `getId`: (`Vértice` $v$) $\rightarrow$ `String`
  Obtiene el identificador del vértice $v$.
- `toString`: (`Vértice` $v$) $\rightarrow$`String`
  Proporciona una representación del vértice $v$ como una cadena de caracteres.

### 1.2. El TAD *Lado*

El **TAD** *Lado* esta formado en su representación por un identificador de tipo `String` y un peso de tipo `double`. Este **TAD** debe ser implementado como una clase abstracta. El **TAD** *Lado* tiene dos subtipos el *Arco* y la *Arista*. Las operaciones de éste **TAD** son las siguientes:

- `crearLado`: (`String` $id$, `double` $p$) $\rightarrow$ `Lado`
  Crea un nuevo lado con un identificador $id$ y un peso $p$.
- `getPeso`: (`Lado` $l$) $\rightarrow$ `double`
  Obtiene el peso del lado $l$.
- `getId`: (`Lado` $l$) $\rightarrow$ `String`
  Obtiene el identificador del lado $l$.
- `toString`: (`Lado` $l$) $\rightarrow$`String`
  Método abstracto para la representación del lado $l$ como una cadena de caracteres.

### 1.2.1. El TAD *Arco*

Subtipo del **TAD** *Lado* que representa a los lados que componen al TAD *Grafo Dirigido*. Es implementado como una clase concreta derivada de la clase abstracta *Lado*. Este **TAD** posee las siguientes operaciones:

- `crearArco`: (`String` $id$, `double` $p$, `Vértice` $vi$, `Vértice` $vf$) → `Arco`
  Crea un nuevo *arco* con un identificador $id$, un peso $p$, un vértice en el extremo inicial $vi$ y un vértice en el extremo final $vf$.
- `getExtremoInicial`: (`Arco` $a$) → `Vértice`
  Obtiene vértice que es el extremo inicial del arco $a$.
- `getExtremoFinal`: (`Arco` $a$) → `Vértice`
  Obtiene vértice que es el extremo final del arco $a$.
- `toString`: (`Arco` $a$) →`String`
  Retorna la representación en `String` del arco $a$.

### 1.2.2. El TAD *Arista*

Subtipo del **TAD** *Lado* que representa a los lados que componen al **TAD** *Grafo No Dirigido*. Es implementado como una clase concreta derivada de la clase abstracta *Lado*. Las operaciones que corresponden al **TAD** *Arista* son las siguientes:

- `crearArista`: (`String` $id$, `double` $p$, `Vértice` $u$, `Vértice` $v$) → `Arista`
  Crea una nueva *arista* con un identificador $id$, un peso $p$, un vértice en el extremo inicial $vi$ y un vértice en el extremo final $vf$.
- `getExtremo1`: (`Arista` $a$) → `Vértice`
  Obtiene vértice que es el primer extremo de la arista $a$.
- `getExtremo2`: (`Arista` $a$) → `Vértice`
  Obtiene vértice que es el segundo extremo de la arista $a$.
- `toString`: (`Arista` $a$) →`String`
  Retorna la representación de la arista $a$ como un `String`.

## 1.3. El TAD *Grafo*

Este **TAD** contendrá las operaciones asociados a un grafo, sea dirigido o no dirigido. Los grafos podrán tener lados múltiples y bucles. El **TAD** *Grafo* debe ser implementado como una interfaz de `JAVA` llamada **Grafo**. Todos los identificadores de los vértices que componen a un grafo deben ser únicos. De la misma manera no deben haber identificadores repetidos de los lados componen a un grafo. Se presentan las operaciones del **TAD** *Grafo* que deben ser implementadas:[1]

- `cargarGrafo`: (`Grafo` $g$, `String` $archivo$) → `boolean`
  Carga en un grafo la información almacenada en el archivo de texto cuya dirección, incluyendo el nombre del archivo, viene dada por $archivo$. El archivo dado tiene un formato determinado que se indicará más abajo. Se retorna `true` si el los datos del archivo son cargados satisfactoriamente en el grafo, y `false` en caso contrario. Este método debe manejar los casos en los que haya problemas al abrir un archivo y el caso en el que el formato del archivo sea incorrecto.

[1] Lista es una colección iterable

- `numeroDeVertices`: (`Grafo` $g$) → `entero`
  Indica el número de vértices que posee el grafo.

- `numeroDeLados`: (`Grafo` $g$) → `entero`
  Indica el número de Lados que posee el grafo.

- `agregarVertice`: (`Grafo` $g$, `Vértice` $v$) → `boolean`
  Agrega el vértice $v$ al grafo $g$ previamente creado. Si en el grafo no hay vértice con el mismo identificador que el vértice $v$, entonces lo agrega al grafo y retorna `true`, de lo contrario retorna `false`.

- `agregarVertice`: (`Grafo` $g$, `String` $id$, `double` $p$) → `boolean`
  Agrega el vértice $v$ al grafo $g$ previamente creado. Si en el grafo no hay vértice con el identificador $id$, entonces se crea un nuevo vértice y se agrega al grafo y se retorna `true`, de lo contrario retorna `false`.

- `obtenerVertice` : (`Grafo` $g$, `String` $id$) → `Vértice`
  Retorna el vértice contenido en el grafo que posee el identificador $id$. En caso que en el grafo no contenga ningún vértice con el identificador $id$, se lanza la excepción `NoSuchElementException`.

- `estaVertice` : (`Grafo` $g$, `String` $id$) → `boolean`
  Se indica si un vértice con el identificador $id$, se encuentra o no en el grafo. Retorna `true` en caso de que el vértice pertenezca al grafo, `false` en caso contrario.

- `estaLado` : (`Grafo` $g$, `String` $u$, `String` $v$) → `boolean`
  Determina si un lado pertenece a un grafo. La entrada son los identificadores de los vértices que son los extremos del lado. En caso de ser aplicada esta función con un grafo dirigido, se tiene que $u$ corresponde al extermo inicial y $v$ al extermo final.

- `eliminarVertice`: (`Grafo` $g$, `String` $id$) → `boolean`
  Elimina el vértice del grafo $g$. Si existe un vértice identificado con $id$ y éste es eliminado exitosamente del grafo se retorna `true`, en caso contrario `false`.

- `vertices`: (`Grafo` $g$) → *Lista de Vertices*
  Retorna una lista con los vértices del grafo $g$.

- `lados`: (`Grafo` $g$) → *Lista de Lados*
  Retorna una lista con los lados del grafo $g$.

- `grado`: (`Grafo` $g$, `String` $id$) → *entero*
  Calcula el grado del vértice identificado por $id$ en el grafo $G$. En caso que en el grafo no contenga ningún vértice con el identificador $id$, se lanza la excepción `NoSuchElementException`.

- `adyacentes`: (`Grafo` $g$, `String` $id$) → *Lista de Vertices*
  Obtiene los vértices adyacentes al vértice identicado por $id$ en el grafo $G$ y los retorna en una lista. En caso que en el grafo no contenga ningún vértice con el identificador $id$, se lanza la excepción `NoSuchElementException`.

- `incidentes`: (`Grafo` $g$, `String` $id$) → *Lista de Lados*
  Obtiene los lados incidentes al vértice identificado por $id$ en el grafo $G$ y los retorna en una lista. En caso que en el grafo no contenga ningún vértice con el identificador $id$, se lanza la excepción `NoSuchElementException`.

- `clone`: (`Grafo` $g$) → `Grafo`
  Retorna un nuevo grafo con la misma composición que el grafo de entrada.

- `toString`: (`Grafo` $g$) → `String`
  Devuelve una representación del contenido del grafo como una cadena de caracteres.

## 1.4. El TAD *Grafo No Dirigido*

Éste **TAD** es una subtipo del **TAD** *Grafo*. Debe ser implementado como una clase concreta que implementa los métodos de la interfaz *Grafo*. El tipo de lado que con el que está constituido esta representación del **TAD** *Grafo*, es la *Arista*. Adicionalmente posee las siguientes operaciones:

- `crearGrafoNoDirigido`: () → `GrafoNoDirigido`
  Crea un nuevo `GrafoNoDirigido`
- `agregarArista` : (`Grafo` $g$, `Arista` $a$) → `boolean`
  Agrega una nueva arista al grafo si el identificador de la arista no lo posee ninguna arista en el grafo. Retorna `true` en caso en que la inserción se lleve a cabo, `false` en contrario.
- `agregarArista` : (`Grafo` $g$, `String` $id$, `double` $p$, `String` $u$, `String` $v$) → `boolean`
  Si el identificador $id$ no lo posee ninguna arista en el grafo, crea una nueva arista y la agrega en el grafo. Retorna `true` en caso en que la inserción se lleve a cabo, `false` en contrario.
- `eliminarArista` : (`Grafo` $g$, `String` $id$) → `boolean`
  Elimina la arista en el grafo que esté identificada con $id$. Se retorna `true` en caso que se haya eliminado la arista del grafo y `false` en caso de que no exista una arista con ese identificador en el grafo.
- `obtenerArista` : (`Grafo` $g$, `String` $id$) `Arista`
  Devuelve la arista que tiene como identificador $id$. En caso de que no exista ninguna arista con ese identificador, se lanza la excepción `NoSuchElementException`.

## 1.5. El TAD *Grafo Dirigido*

Éste **TAD** es una subtipo del **TAD** *Grafo*. Debe ser implementado como una clase concreta que implementa los métodos de la interfaz *Grafo*. El tipo de lado que con el que está constituido el Digrafo, es el *Arco*. Adicionalmente posee las siguientes operaciones:

- `crearGrafoDirigido`: () → `GrafoDirigido`
  Crea un nuevo `GrafoDirigido`
- `agregarArco` : (`Grafo` $g$, `Arco` $a$) → `boolean`
  Agrega un nuevo arco al grafo si el identificador del arco no lo posee ningún arco en el grafo. Retorna `true` en caso en que la inserción se lleva a cabo, `false` en contrario .
- `agregarArco` : (`Grafo` $g$, `String` $id$, `double` $p$, `String` $eInicial$, `String` $eFinal$) → `boolean`
  Si el identificador $id$ no lo posee ningún arco en el grafo, crea un nuevo arco y lo agrega en el grafo. Retorna `true` en caso en que la inserción se lleva a cabo, `false` en contrario .
- `eliminarArco` : (`Grafo` $g$, `String` $id$) → `boolean`
  Elimina el arco en el grafo que esté identificado con $id$. Se retorna `true` en caso que se haya eliminado el arco del grafo y `false` en caso que no exista un arco con ese identificador en el grafo.
- `obtenerArco` : (`Grafo` $g$, `String` $id$) `Arco`
  Devuelve el arco que tiene como identificador $id$. En caso de que no exista ningún arco con ese identificador, se lanza la excepción `NoSuchElementException`.
- `gradoInterior` : (`Grafo` $g$, `String` $id$) → `entero`
  Calcula el grado interior del vértice identificado por $id$ en el grafo. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.

- `gradoExterior` : (`Grafo` $g$, `String` $id$) $\rightarrow$ `entero`
  Calcula el grado exterior del vértice identificado por $id$ en el grafo. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.

- `sucesores`: (`Grafo` $g$, `String` $id$) $\rightarrow$ *Lista de Vértices*
  Devuelve una lista con los vértices que sucesores del vértice con identificador $id$. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.

- `predecesores`: (`Grafo` $g$, `String` $id$) $\rightarrow$ *Lista de Vértices*
  Devuelve una lista con los vértices predecesores del vértice con identificador $id$. En caso de que no exista ningún vértice con ese identificador, se lanza la excepción `NoSuchElementException`.

# 2. Detalles de la implementación

Debe implementar los **TAD**s *Grafo Dirigido* y *Grafo No Dirigido*, como listas de adyacencias. Se provee de un conjunto de archivos con las firmas cuya implementación debe ser completada. Los archivos son: `Lado.java`, `Arco.java`, `Arista.java`, `Vertice.java`, `Grafo.java`, `Digrafo.java`, `GrafoNoDirigido.java` y `ClienteGrafo.java`. El archivo `ClienteGrafo.java` tiene como objetivo servir como un cliente en el cual se puedan probar las funcionalidades de los **TAD**s. Sus implementaciones de los operadores deben ser *razonablemente eficientes*. Todo el código debe estar debidamente documentado. Se deben indicar una descripción del método, la descripción de los parámetros de entrada y salida, las pre y post condiciones y el orden de ejecución de cada método aplicando el estándar para la documentación de código en `JAVA`. Su implementación debe incluir manejo de excepciones. Puede usar las librerías de `JAVA` que considere útiles. Su código debe hacer uso de la guía de estilo publicada en el Aula Virtual.

El formato del archivo que contiene los datos de un grafo es el siguiente:

`NumeroDeVertice`

`NumeroDeLados`

$\texttt{idVertice}_1$ $\texttt{pesoVertice}_1$

...

...

$\texttt{idVertice}_{NumeroDeVertice}$ $\texttt{pesoVertice}_{NumeroDeVertice}$

$\texttt{idLado}_1$ $\texttt{idVerticeInicial}_1$ $\texttt{idVerticeFinal}_1$ $\texttt{pesoLado}_1$

...

...

$\texttt{idLado}_{NumeroDeLados}$ $\texttt{idVerticeInicial}_{NumeroDeLados}$ $\texttt{idVerticeFinal}_{NumeroDeLados}$ $\texttt{pesoLado}_{NumeroDeLados}$

En la evaluación del proyecto se tomará en cuenta aspectos como la documentación, el estilo de programación, la modularidad del código, la eficiencia en tiempo de ejecución y memoria, el uso de herencia, el manejo de excepciones, el buen uso de las librerías y la robustez. Usted debe realizar los casos de pruebas que muestren el correcto funcionamiento de las funciones implementadas. Si todos los archivos fuentes del proyecto no compilan correctamente, el proyecto será calificado con cero.

## 3. Condiciones de la entrega

La entrega en digital del proyecto es hasta el miércoles de la semana 5 (12 de Octubre) a la 1:30 pm. La entrega del sobre será en la oficina de la profesora Martínez (MYS 229A) el Jueves 13 de Octubre, antes de la 1:30 pm. Su entrega debe incluir lo siguiente:

- Un sobre sellado y debidamente identificado con sus nombres, carnés y profesor de laboratorio. Éste debe contener estos tres elementos:
  - El código de su proyecto, debidamente documentado.
  - Un reporte de no más de dos páginas en donde se explique el diseño de su solución y se indiquen los detalles más relevantes de la implementación realizada.
  - La "Declaración de Autenticidad para Entregas" firmada por los autores del proyecto.
- Un archivo comprimido del tipo `TGZ` con el código fuente de su proyecto, que debe ser entregado en la página del curso en el Aula Virtual. El nombre del archivo deber ser **Proy1ci2693SepDic16-X-Y.tgz** donde **X** y **X** son los números de carné de los autores del proyecto.

El no cumplimento de todos los requerimientos podría resultar en el rechazo de su entrega.

---

Guillermo Palma / gvpalma@usb.ve / Septiembre 2016