## Especificación clases compartidas

**Nom**: Graph

**Descripció**: Classe encarregada de gestionar tots els elements que conformen un graf, és a dir, tots els nodes i les seves relacions.

**Cardinalitat**: El controlador del domini sempre tindrà una única instància de graph (el graf que tingui l'usuari obert en aquell moment).

**Atributs**: name:String, nodeContainers:HashMap<NodeType, Container<Node>>, relations:Container<Relation>

**Metodes:**

* **public Graph**(String name): Constructor de la classe Graph. Instància un container per cada tipus de node i afegeix les relacions bàsiques.
* **public Graph**(): Construeix el Graph passant com a nom "-1".
* **public String getName**(): Retorna el nom del graf.
* **public void setName**(String name): Edita el nom del graf
* **public int getSize**(NodeType type): Retorna el nombre de nodes del tipus indicat que hi ha al graf.
* **public void addNode**(Node node): Afegeix un node al graf. Mira de quin tipus és el node per saber a quin container introduir-lo. Es fa servir per nodes nous introduits manualment per l'usuari (l’Id serà el que toqui per defecte).
* **public void addNode**(Node node, int id): Afegeix un node al graf. Mira de quin tipus és el node per saber a quin container introduirlo. Es fa servir per nodes llegits a la capa de persistencia (ja sabem el seu id).
* **public Node createNode**(NodeType type, String value): Classe encarregada de crear un nou node.
* **public void removeNode**(NodeType type, int id): Classe encarregada d’eliminar un node.
* **public Node getNode**(NodeType type, int id): Donat un tipus de node y el seu Id, retorna el respectiu node.
* **public ArrayList<Node> getEdges**(int relationID, Node node): Retorna una llista dels nodes als cuals està connectat el node paràmetre.
* **public ArrayList<Node> getEdges**(int relationID, NodeType type, int id): Igual que el metode anterior pero passant el node en forma de Id+TipusNode en comptes d’una instància de Node.
* **public void addEdge**(int relationID, Node a, Node b): Donats dos nodes diferents afegeix una aresta que els connecti.
* **public void addEdge**(int relationID, NodeType typeA, int idA, NodeType typeB, int idB): Fa el mateix que el mètode anterior pero en comptes de rebre els nodes paràmetre com una instància de node, els reb com a parelles Id+TipusNode.
* **public void removeEdge**(int relationID, Node a, Node b): Donats dos nodes esborra la aresta que els connecta.
* **public void removeEdge**(int relationID, NodeType typeA, int idA, NodeType typeB, int idB): Fa el mateix que el mètode anterior pero en comptes de rebre els nodes paràmetre com una instància de node, els reb com a parelles Id+TipusNode.
* **public Container<Node>.ContainerIterator getNodeIterator**(NodeType type): Retorna un iterador que permet recorrer tots els nodes del tipus passat com a paràmetre.
* **private void i\_addRelation**(Relation relation, NodeType type)
* **public void addRelation**(Relation relation): Afegeix una relacio nova al sistema. Part del codi d’aquest mètode està al mètode privat i\_addRelation.
* **private void i\_removeRelation**(Relation relation, NodeType type)
* **public void removeRelation**(int relationID): Elimina una relació existent. Part del codi d’aquest mètode està al mètode privat i\_addRelation.
* **public Relation getRelation**(int relationID): Retorna la relació referenciada per l’Id passat com a paràmetre.
* **private NodeType getNodeType**(Node node): Classe encarregada de retornar el tipus del node passat com a paràmetre.
* **private Container<Node> getNodeContainer**(NodeType type): Retorna el contenidor associat als nodes del tipus indicat.

**Nombre:** Container

**Descripción:** Representa un conjunto de elementos.

**Cardinalidad:** 1

**Atributos:**

· lastID (representa el último id asignado).

· elements (HashMap que contiene el conjunto de elementos identificados por un id)

**Relaciones:** Dos asociaciones con Graph.

**Métodos:**

* **Container**()

Constructor de la clase que inicializa los atributos.

* **getSize**()

Devuelve el tamaño de elements.

* **getLastID**()

Devuelve lastID.

* **getIterator**()

Devuelve un iterador de elements.

* **addElement**(T element)

Añade element al hashmap con lastID como identificador.

* **addElement**(T element, int ID)

Añade element al hashmap con ID como identificador.

* **removeElement**(int ID)

Elimina el elemento con identificador ID del hashmap.

* **getElement**(int ID)

Devuelve el elemento del hashmap con identificador ID.

* **getIDFromValue**(String value)

Devuelve el identificador del elemento con valor value.

* **checkID**(int ID)

Devuelve true si el ID pasado como parámetro es válido.

* **checkNewID**(int ID)

Devuelve true si el ID pasado como parámetro podría ser el identificador de un nuevo elemento.

**Nombre:** AuthorLabel

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Author y otra de tipo Label.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* AuthorLabel()

Constructor de la clase que crea una relación entre un Author y un Label.

**Nombre:** AuthorPaper

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Author y otra de tipo Paper.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* **AuthorPaper()**

Constructor de la clase que crea una relación entre un Author y un Label.

**Nombre:** ConferenceLabel

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Conference y otra de tipo Label.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* **ConferenceLabel()**

Constructor de la clase que crea una relación entre una Conference y un Label.

**Nombre:** ConferencePaper

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Conference y otra de tipo Paper.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* **ConferencePaper()**

Constructor de la clase que crea una relación entre una Conference y un Paper.

**Nombre:** PaperLabel

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Paper y otra de tipo Label.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* **PaperLabel()**

Constructor de la clase que crea una relación entre un Paper y un Label.

**Nombre:** TermPaper

**Descripción:** Subclase de la clase Relation que representa la relación entre una entidad de tipo Term y otra de tipo Paper.

**Cardinalidad:** 1. Se instanciará que se cree una relación de este tipo.

**Atributos:** No tiene

**Relaciones:**

* Asociación con la clase Graph. Graph crea a esta clase.

**Métodos:**

* **TermPaper()**

Constructor de la clase que crea una relación entre una Term y un Paper.

**Nombre:** GraphSearch

**Descripción:** Clase abstracta que representa una búsqueda en la base de datos (representada en el grafo). Se instanciarán sus subclases dependiendo del tipo de búsqueda. Extraerá una lista de elementos que cumplen con las condiciones de búsqueda.

**Cardinalidad:** Al ser abstracta, 0. Sus subclases se instanciarán una cada vez que se realice una búsqueda.

**Atributos:**

* results (arrayList de Result que contiene el resultado de una búsqueda).
* graph (grafo sobre el que se realiza la búsqueda).
* executed (booleano que indica si se ha ejecutado una búsqueda)

**Relaciones:**

* Asociación con la clase Grafo: Indica sobre qué datos se realiza la búsqueda.

**Métodos:**

* **GraphSearch(Graph graph)**

Constructor de la clase que obtiene el grafo sobre el que se realizarán las búsquedas

* **search()**

Realiza la búsqueda

* **getResults()**

Devuelve una lista de los resultados de la búsqueda.

**Nombre:** Result

**Descripción:** Subclase de la clase GraphSearch. Representa uno de los resultados de una búsqueda.

**Cardinalidad:** Muchos, una búsqueda puede tener muchos resultados

**Atributos:**

* from (Nodo a partir del qual se ha realizado la busqueda, no static)
* to (Nodo encontrado al realizar la busqueda, no static)
* hetesim (valor resultante al aplicar el algoritmo HeteSim, no static)

**Relaciones:**

* Asociación con la clase Nodo.

**Métodos:**

* **Result(Node from, Node to, double hetesim)**

Constructor de la clase que obtiene todos los atributos

**Nombre:** Búsqueda simple(SimpleSearch)

**Descripción:** Subclase de la clase GraphSearch. Representa una búsqueda simple, es decir, una búsqueda en la que se especifica un tipo de nodo del grado y un filtro. Muestra los nodos del tipo especificado que contienen el filtro de búsqueda.

**Cardinalidad:** 1. Se instanciará cada vez que se realice una búsqueda simple.

**Atributos:**

* filter (Filtro con el que se realiza la búsqueda, no static).
* type (tipo de nodo sobre el que se realiza la búsqueda, no static).

**Relaciones:**

* Asociación con la clase Nodo (en realidad, subclases de Nodo). Indica qué tipos de elementos se van a buscar. Ej: autores.

**Métodos:**

* **SimpleSearch(string filtro)**

Constructor de la clase que obtiene el grafo sobre el que realizar la busqueda, el filtro i el tipo de nodo.

* **search()**

Método que realiza la búsqueda.

**Nombre:** Búsqueda de relaciones(RelationalSearch)

**Descripción:** Representa una búsqueda que sigue una estructura de relación determinada. Dependiendo de si se especifica un nodo de origen y destino, tenemos otras 3 subclases.

**Cardinalidad:** Al ser abstracta, 0.

**Atributos:**

* **rs** (estructura de relación de la que se tiene que realizar la búsqueda, no static).
* **EdgesID**(hashmap para relacionar los nodos con sus IDs, no static).
* **FirstMatrix**(variable auxiliar para saber si se ha creado ya la primera matriz, no static).
* **Free\_j\_edgeMatrix**(variable para saber qué columnas quedan libres al hacer una búsqueda con nodo de origen en la cual la primera relación del RelationStructure es una edgeRelation(tiene nodos imaginarios en el medio). Es una variable para un cálculo muy particular, no static).

**Relaciones:**

* Dependencia con la clase Matrix. La utilizará para calcular el hetesim entre los elementos del grafo.
* Asociación con la clase Estructura de relaciones. Será lo que defina qué estructura de relación seguirá al buscar los elementos.

**Métodos:**

* **RelationalSearch**(Graph graph, RelationStructure rs)

Constructor de una búsqueda relacional, a la que se le pasa la estructura de relaciones y el grafo donde realizar la búsqueda.

* **Search**()

Método que realiza la búsqueda.

* **Hetesim**()

Retorna la matriz con los valores de hetesim.

* **i\_hetesim**(int i, int j, boolean normalize\_rows)

Método auxiliar (función de inmersión) para los cálculos de hetesim.

* **getNormalizedMatrix**(Relation relation, boolean normalize\_rows)

Devuelve la matriz normalizada a partir de la relación.

* **getNormalizedEdgeLeftMatrix**(Relation edgeRelation)

Devuelve la matriz normalizada de la parte izquierda de la edgeRelation (relación del medio con nodos imaginarios).

* **getNormalizedEdgeRightMatrix**(Relation edgeRelation)

Devuelve la matriz normalizada de la parte izquierda de la edgeRelation (relación del medio con nodos imaginarios).

* **getColumnsEdgeMatrix**(Relation edgeRelation)

Devuelve las columnas/filas de las matrices normalizadas de la edgeRelation

* **generateResults**(Matrix matrix)

Método abstracto que tienen que implementar las subclases para seleccionar y presentar los resultados

* **createFirstMatrix**(Relation relation)

Método abstracto que tienen que implementar las subclases para crear la primera matriz, ya que para optimizar las búsquedas con nodo fijo, no se realiza el cálculo del resto de nodos

* **createFirstEdgeMatrix**(Relation edgeRelation)

Método abstracto que tienen que implementar las subclases para crear la edgeMatrix en el caso que sea la primera matriz.

* **getIteratorFirstMatrix**(Matrix matrix)

Método abstracto que tienen que implementar las subclases para iterar sobre la primera matriz

**Nombre:** Búsqueda libre(FreeSearch)

**Descripción:** Subclase de la clase Búsqueda por relaciones. Representa una búsqueda en la que no se especifica desde qué elemento sale la relación definida por una Estructura de relaciones. Buscará todos los elementos del último tipo de la estructura relacionados con todos los elementos del primero.

**Cardinalidad:** 1. Se instanciará cada vez que se realice una búsqueda libre.

**Atributos:** Ninguno.

**Relaciones:**

* Dependencia de Matrix.

**Métodos:**

* **FreeSearch**(Graph graph, RelationStructure rs)

Constructor de una búsqueda libre, a la que se le pasa la estructura de relaciones y el grafo donde realizar la búsqueda.

* **getIteratorFirstMatrix**(Matrix matrix)
* **createFirstMatrix**(Relation relation)
* **createFirstEdgeMatrix**(Relation edgeRelation)

**Nombre:** Búsqueda origen(OriginSearch)

**Descripción:** Subclase de Búsqueda por relaciones. Representa una búsqueda en la que se especifica un nodo de origen. Esto significa que buscará todos los nodos que estén relacionados con el de origen que cumplan con la Estructura de relaciones y condiciones de la búsqueda.

**Cardinalidad**: 1. Se instanciará cada vez que se realice una búsqueda con origen fijado.

**Atributos:**

* Node from(nodo origen de la búsqueda relacional, no static)

**Relaciones:**

* Asociación con la clase Nodo. Indica de qué instancia parte la Estructura de relaciones.

**Métodos:**

* **OriginSearch**(Graph graph, RelationStructure rs, Node From)

Constructor de una búsqueda con nodo origen establecido, a la que se le pasa la estructura de relaciones, el grafo donde realizar la búsqueda y el nodo origen de la búsqueda relacional.

* **getIteratorFirstMatrix**(Matrix matrix)
* **createFirstMatrix**(Relation relation)
* **createFirstEdgeMatrix**(Relation edgeRelation)

**Nombre:** Búsqueda origen destino(OriginDestinationSearch)

**Descripción:** Subclase de Búsqueda por relaciones. Representa una búsqueda en la que se especifica un nodo de origen y otro de destino. Básicamente encuentra el nivel de relación entre los dos elementos especificados (HeteSim) en el grafo en el que se busca.

**Cardinalidad:** 1. Se instanciará cada vez que se realice una búsqueda con origen y destino fijados.

**Atributos:**

* Node from (nodo origen de la estructura de relación,no static)
* Node to (nodo destino de la estrucutra de relación, no static).

**Relaciones:**

* Asociación con la clase Nodo. Indica de qué instancia parte la Estructura de relaciones.
* Asociación con la clase Nodo. Indica en qué instancia termina la Estructura de relaciones.

**Métodos:**

* **BúsquedaOrigenDestino**(Graph graph, RelationStructure rs, Nodo from, Nodo to)

Constructor de una búsqueda con nodo origen y destino establecidos, a la que se le pasa la estructura de relaciones, el grafo donde realizar la búsqueda y el nodo origen y destino de la búsqueda relacional.

* **generateResults**(Matrix matrix)

**Nombre:** Estructura de relaciones(RelationalStructure)

**Descripción:** Clase que representa una secuencia de tipos de nodos y relaciones en la que se basará la mayoría de tipos de búsqueda. Será la forma de especificar la estructura de relaciones del grafo que se quiere contemplar en una búsqueda por relaciones.

**Cardinalidad:** 1 por búsqueda por relaciones.

**Atributos:** Ninguno

**Relaciones:**

* Agregación con la clase Relación. Las relaciones agregadas y su ordenación indicará la estructura con la que se realizará una búsqueda.

Llama al constructor de la super clase pasándole la estructura y guarda desde qué nodo sale y acaba la búsqueda.

**Métodos:**

* **RelationStructure**(NodeType From, ArrayList<Relation> r, NodeType to)

Constructor de la clase. Inserta el tipo "nulo" en medio de las estructuras pares.

* **RelationStructure**(Graph graph, NodeType from, int[] ids, NodeType to)

Constructor de la clase. Usa el primer constructor, transformando els grafo y las ids en una arraylist de relaciones.

* **cloneArrayList**(ArrayList<Relation> r)

Método que sirve para clonar una arraylist.

* **setOrder**(NodeType from, int i, NodeType to)

Método para comprobar si el orden de una relación es el correcto para poder hacer hetesim posteriormente, si no es el correcto se encarga de corregirlo.

* **getArrayList**(Graph graph, int[] ids)

Devuelve una arraylist de las relaciones que tienen los ids en el grafo, ambos son párametros necesarios de la función.

**Nombre:** Matriz(Matrix)

**Descripción:** Clase que se utiliza para el cálculo de HeteSim, a partir de matrices de adyecendcia.

**Cardinalidad:** 1 por cada vez que se calcule HeteSim.

**Atributos:**

* rows(arraylist de las filas de la matriz, no static)
* euclidean(array que mantiene la media euclidiana de las columnas, para optimizar su cálulo al normalizar la matriz)
* n(número de filas de la matriz)
* m(número de columnas de la matriz)

**Relaciones:** Ninguna

**Métodos:**

* **Matrix**(int n, int m)

Constructor de la clase. Crea una matriz n\*m.

* **Get**(int i, int j)

Nos devuelve el valor de la posición (i,j).

* **Put**(int i, int j, double value)

Pone el valor (value), en la posición (i,j) de la matriz.

* **Mul**(Matrix mat)

Retorna la matriz resultante de multiplicar dos matrices.

* **GetRows**()

Retorna el numero de filas de la matriz.

* **GetColumns**()

Retorna el numero de columnas de la matriz.

* **i\_mul**(Matrix mat, boolean normalize)

Método auxiliar de mul.

* **getEuclidean**(int j)

Devuelve la media euclidiana de la columna j.

* **mul\_norm**(Matrix mat)

Retorna la matriz hetesim normalizada resultante de multiplicar dos matrices.

* **binarySearch**(ArrayList<Cell> row, int j)

Realiza una búsqueda binaria para encontrar la celda de la columna j, si existe. Devuelve false si no existe.

* **i\_binarySearch**(ArrayList<Cell> row, int l, intr, int j)

Función de inmersión de binarysearch.

* **checkPos**(int i, int j)

Método que retorna si una posición es válida.

**Nombre:** Estructura de relaciones exception(RelationalStructureException)

**Descripción:** Clase para controlar los errores de la estrucutura de relación.

**Cardinalidad:** 1 por cada relación de estrucutura que se crea.

**Atributos:**

* Error (error que lanzamos cuando de produce un fallo, no static).

**Relaciones:** Ninguna

**Métodos:**

* **RelationStructureExcpetion**(Error error)

Constructor de la clase.

* **GetError**()

Devuelve el error provocado.

* **PrintStackTrace**()

Método que lanza mensajes de error, dependiendo que que tipo de error lanza un mensaje distinto.

**Nom**: NodeType (Domini)

**Descripció:** Classe de tipus Enumeration que defineix un dels 5 tipus de Nodes que hi ha al programa (Paper, Author, Conference, Label, Term).

**Cardinalitat:** -

**Atributs**: -

**Mètodes: -**

**Nom**: Node (Domini)

**Descripció**: Classe que s’encarrega de crear les instàncies de nodes, i tenir totes les relacions que té.

**Cardinalitat**: Molts. Un graf tindrà moltes instàncies de la classe Node

**Atributs**: private HashMap<Integer, HashSet<Integer>> relations;

**Metodes:**

* **protected Node(String value):** Crea un nou node donat el seu nom.
* **public Node(int nodeID, String value):** Crea un nou node donant-ne el seu identificador i el seu nom.
* **public Node(Node n) :** Crea una copia del node.
* **protected void addEdge(int relationID, int nodeID):** Afegeix una relació del tipus donat per el primer parametre amb el node determinat per el segon.
* **protected void removeEdge(int relationID, int nodeID):**  Esborra una relació del tipus donat per el primer parametre amb el node determinat per el segon.
* **public boolean isRelated(int relationID, int nodeID):** Determina si un node està relacionat amb un altre donant el id de la relació i el id del segon node.
* **public boolean isRelated(int relationID, Node node):** Determina si un node està rlacionat amb un altre donant el id de la relació i el segon node.
* **protected ArrayList<Integer> getEdges(int relationID):** Retorna totes les relacions del tipus relationID que té el node.
* **public int getSizeEdges(int relationID):** Retorna el nombre de relacions del tipus relationID que té el node.
* **protected void addRelation(Relation relation):**  Afegeix un nou tipus de relació per al node.
* **protected void removeRelation(Relation relation):**  Esborra un tipus de relació per al node.

**Nom**: Relation (Domini)

**Descripció**: Classe que representa un tipus de relació entre dos tipus de nodes. És subclasse d’Element.

**Cardinalitat**: Moltes. S’instanciarà un cop per cada tipus de relació present al graf.

**Atributs**: NodeType A, Nodetype B, String name, Int Id.

**Metodes:**

* **public Relation**(NodeType A, NodeType B, String name): constructora que crea una instància de Relation, que va d’un node de sortida A a un node d’entrada B i que té nom name.
* **public Relation**(NodeType A, NodeType B, String name, int id): constructora que crea una instància de Relation, que va d’un node de sortida A a un node d’entrada B, que té nom name i identificador igual a id.
* **public Relation**(Relation r): constructora que crea una instancia de Relation, que serà copia de la relació “r” passada per paràmetre.
* **public String getName**() : retorna el nom de la relació.
* **protected boolean isDefault**(): retorna true si la relació és una de les relacions predefinides. Retorna false altrament.
* **protected NodeType getNodeTypeA**(): retorna el tipus de node del node de sortida de la relació.
* **protected void setNodeTypeA**(Nodetype type): canvia el tipus de node del node de sortida de la relació per l’indicat al paràmetre type.
* **protected NodeType getNodeTypeB**(): retorna el tipus de node del node d’entrada de la relació.
* **protected void setNodeTypeB**(Nodetype type): canvia el tipus de node del node d’entrada de la relació per l’indicat al paràmetre type.

**Nom**: Element (Domini)

**Descripció**: Classe abstracta dedicada a representar qualsevol element dins del domini.

**Cardinalitat**: 0. És abstracta.

**Atributs**: String: value, int: id

**Metodes:**

* **Element(String Value):** Constructora.
* **Element(int Id, String Value):** Constructora.
* **Element (Element e):** Constructora.
* **getId():** retorna la id de l’element
* **setId(int id):** assigna un id a l’ element.
* **setValue(String Value):** guarda la informació a l’element.
* **getValue():** retorna la informació d’un element

**Nom: PersistenceController**

**Descripció:** Capa intermèdia entre la memòria i el disc. Es comunica amb la resta del programa per a la importació/exportació de dades i utilitza els serialitzadors per a convertir-les.

**Cardinalitat:** 1 per sessió del programa

**Atributs:** Grago graph

**Relacions:** Agregació amb la classe Serialitzador Nodes y Serialitzador Arestes

**Mètodes:**

* PersistenceController(Graf graph)

Constructor inicial de la classe, que la associa a un graf

* Void importNodes(String path, NodeType type)

Mètode per a importar els nodes, de un fitxer txt

* Void importEdges(String path, NodeType type1, NodeType type2)

Mètode per a importar les relacions, d’un fitxer txt

* exportGraph(String path)

Mètode per a guardar el graf amb tots els seus nodes i relacions en fitxers txt

**Nom: Serialitzador Node**

**Descripció:** Serialitza els nodes. És capaç de serialitzar qualsevol tipus de node.

**Cardinalitat**: Molts. S’instanciarà un per cada fitxer a importar/exportar.

**Atributs**: Node node, String data, int nodeId, String name

**Relacions:** Cap

**Mètodes:**

* NodeSerializer(Node node)

Constructor de la classe a partir d’un node

* NodeSerializer(String data)

Constructor de la classe a partir d’un String

* Void inflate()

Mètode per a importar un node a partir d’un String

* Void deflate()

Mètode per a exportar el node en un String

* String getData()

Mètode per a o**b**tenir el String d’un node

* int getId()

Mètode per a obtenir el ID d’un node

* String getName()

Mètode per a obtenir el nom de l’objecte (Author,paper,conference,term)

**Nom**: **Serialitzador Aresta**

**Descripció:** Serialitza les relacions "arestes". És capaç de serialitzar tant els tipus predeterminats com els especificats per els usuaris.

**Cardinalitat**: Molts. S’instanciarà un per cada fitxer a importar/exportar

**Atributs**: String data, Graph graph, NodeType ntype1, NodeType ntype2, Node node1, Node node2

**Relacions:** Cap

**Mètodes:**

* EdgeSerializer(Graph graph, String data, NodeType ntype1, NodeType ntype2)

Constructor de la classe a partir d’un graf, el String i dos tipus de node

* EdgeSerializer(Node node1, Node node2)

Constructor de la classe a partir de dos nodes

* Void inflate()

Mètode para importar el String, y obtenir dos Object node

* Void deflate()

Mètode per a exportar los dos Objects en un String

* Node getNode1()

Mètode per a obtenir el primer node de la relació

* Node getNode2()

Mètode per a obtenir el segon node de la relació

* String getData()

Mètode per a obtenir el String dels dos nodes relacionats

**Nom**: **Serialitzador Etiqueta**

**Descripció:** Implementació específica de la relació Aresta que permet llegir/escriure les etiquetes.

**Cardinalitat**: Molts. S’instanciarà un per cada fitxer a importar/exportar

**Atributs**: Cap

**Relacions:** Cap

**Mètodes:**

* LabelSerializer(Graph graph, String data, NodeType ntype1, NodeType ntype2)

Constructor de la classe a partir d’un graf, el String y els dos tipus de nodes del String

* LabelSerializer(Node node1, Node node2)

Constructor de la classe a partir de dos nodes.

* Void inflate()

Mètode per a importar el String, y obtenir dos Object nodo

* Void deflate()

Mètode per a exportar els dos Objects en un String