Each time you add a node or edge in the graph, a corresponding object is created for you. These objects belong to the Node and Edge interfaces. As you have seen it above, you name the nodes and edges by an identifier under the form of a string of characters. Each identifier must be unique since they relate to only one node or one edge.

Once added in a graph with an identifier, you can obtain a reference on these objects using this identifier:

Node A = graph.getNode("A");

or:

Edge AB = graph.getEdge("AB");

The Node and Edge interfaces then allow to obtain details on the represented nodes and edges. For example:

* what is the unique identifier of the node or edge (Node.getId(), Edge.getId()),
* what is the degree of the node (number of connected edges, Node.getDegree()),
* what are the edges connected to a node (Node.hasEdgeToward(String id) and Node.getEdgeToward(String id)),
* is an edge directed or not (Edge.isDirected()),
* what is the node at the other end of the edge (Edge.getOpposite(Node n)),
* etc.

In addition to getNode and getEdge which impose to know the identifier of the node or edge, the graph class provides ways to iterate on the nodes or edges, you can write:

for(Node n:graph) {

System.out.println(n.getId());

}

The code above allows to iterate on all the nodes of the graph and will print their unique identifier, one by line. You can do a similar thing for edges:

for(Edge e:graph.getEachEdge()) {

System.out.println(e.getId());

}

In fact the for(Node n:graph) instruction is a shorthand for:

for(Node n:graph.getEachNode()) {

...

}

You can also obtain a read-only set of nodes from the graph (this is not a copy, but a view on the set of nodes, hence the operation is reasonably fast):

Collection<Node> nodes = graph.getNodeSet();

This can be useful since a lot of algorithms in the java.util.Collections are then applicable on the obtained set. The same operation is possible on edges using getEdgeSet().

Finally, you can also, if you prefer, use iterators on theses sets of nodes and edges:

Iterator<? extends Node> nodes = graph.getNodeIterator();

while(nodes.hasNext()) {

Node node = nodes.next();

...

}

And anew, this is exactly the same for edges with Graph.getEdgeIterator().

Another way to access graph elements is to use their indices. Unlike identifiers which are specified when the elements are created, indices are automatically maintained by the graph. They could change when elements are added or removed but are always between zero and node/edge count minus one. The following loop iterates on all the nodes of the graph:

for (int i = 0; i < graph.getNodeCount(); i++) {

Node node = graph.getNode(i);

...

}

Access by index is generally faster than access by identifier. It can be useful to interface GraphStream with APIs that use arrays. The following code constructs the adjacency matrix of a graph:

int n = graph.getNodeCount();

byte adjacencyMatrix[][] = new byte[n][n];

for (int i = 0; i < n; i++)

for (int j = 0; j < n; j++)

adjacencyMatrix[i][j] = graph.getNode(i).hasEdgeBetween(j) ? 1 : 0;

To get the index of a node knowing its identifier, use:

int i = graph.getNode("A").getIndex();

Inversely, to get the identifier of a node when its index is known, use

String id = graph.getNode(0).getId();