Weighted Directed Graph Java implemenation document

Design details:

- **The inbound and outbound nodes were kept using LinkedHashMaps between an int representing the vertex and a list of adjacent nodes**
- **The weights also, with a LinkedHashMap between an Edge and the weight (int).**

Meaningful interface of classes created:

public class graph.**DirectedGraph** {
Create a DirectedGraph from maps of in and out nodes given
public graph.DirectedGraph(Map<Integer, List<Integer>>, Map<Integer, List<Integer>>);
Create a DirectedGraph with empty maps
public graph.DirectedGraph();
Deep copy of self
public graph.DirectedGraph copy();
Load graph from file on path into self
public void loadGraphFromFile(Path);
Get number of vertices of self
public int getNumberOfVertices();
Get iterator over the set of vertices
public Iterator getSetOfVerticesIterator();
Check whether there is an edge between given vertices
public boolean hasEdgeBetween(int, int);
Get edge between given vertices. If not found, throw NPE
public graph.Edge getEdgeBetween(int, int) throwsNullPointerException;
Get in degree of given vertex. If not found, throw NPE
public int getInDegreeOfNode(int) throws NullPointerException;
Get out degree of given vertex. If not found, throw NPE
public int getOutDegreeOfNode(int) throws NullPointerException;
Get outbound edges iterator. If vertex not found, throw NPE
public Iterator getOutboundEdgesOfNodeIterator(int) throws NullPointerException;
Get inbound edges iterator. If vertex not found, throw NPE
public Iterator getInboundEdgesOfNodeIterator(int) throws NullPointerException;
Add given edge. If existing, throw DuplicateEdgeException
public void addEdge(graph.Edge) throws DuplicateEdgeException;
Remove edge if existing, else throw NPE
public void removeEdge(graph.Edge) throws NullPointerException;
Add node if not existing, else throw NodeAlreadyExistingException
public void addNode(int) throws NodeAlreadyExistingException;
Remove node if existing, else throw NPE
public void removeNode(int) throws NullPointerException;
Get number of edges of self
public int getNumberOfEdges();

}

public class graph.**WeightedDirectedGraph** extends graph.DirectedGraph {
Create a DirectedGraph from maps of in and out nodes given, plus the weights as map
  public graph.WeightedDirectedGraph(Map<Integer, List<Integer>>, Map<Integer, List<Integer>>, Map<Edge, Integer>);
Get edge of weight if existing, throw NPE otherwise
  public int getWeight(graph.Edge);
Update edge weight of edge existing, throw NPE otherwise
  public void updateWeight(graph.Edge, int);
}

Menu (command - explanation):

1 - number of vertices
2 - set of vertices
3 e1 e2 - edge between e1 and e2
4 v - in degree of v
5 v - out degree of v
6 v - outbound edges of e
7 v - inbound edges of e
8 e - weight of e
9 e w - add edge e with weight w
10 e - remove edge e
11 v - add vertex v
12 v - remove vertex v
13 - copy current graph
14 e w - update edge e to weight w