The templates required for this exercise can be downloaded here:
https://www.dfki.de/~steffen/advanced-java/graphs.zip

1 Reading Graph from String (4 points)

Implement methods to create a graph from text based specifications.

The first specification consists of lines containing a single start vertex label followed by an arrow followed by one or more end vertex labels. For such a line, create separate edges from the start vertex to each of the end vertices. Each edge should have as edge info a string created by concatenating the start and end vertex labels.
Example:

```
A --> B E
B --> C E
C --> D
D --> B
E --> D F
F --> D
G --> H I
H --> A E
I --> G H
```

The second specification additionally contains edge weights in round brackets after each end vertex. The edge weight should replace the default edge info string from the first specification with the weight as integer.
Example:

```
s --> w(2) z(4)
z --> w(5) y(9)
v --> w(1) s(6)
w --> x(6) q(3)
t --> u(1) v(9) s(8)
u --> t(5) v(7)
x --> z(8) u(7)
q --> x(2)
y --> x(5)
r --> s(1)
```

Implement both methods by filling the stub methods `readGraph` and `readGraphWithWeights` in the provided `DirectedGraph` class.

Write unit tests to verify that the graphs you create are correct. Hint: The graph's `toString` method returns a list of all edges with their start and end vertices and labels, but don't rely on the order!

2 Breath/Depth-First-Search Visitors (6 points)

Implement the method stubs `bfsVisit` and `dfsVisit` in the provided `DirectedGraph` class so that the graph is traversed in BFS/DFS order. Make sure that the methods of the visitor argument are called in the right places. You don't have to write unit tests for this task.

3 Timestamps Visitor (5 points)

Implement a DFS visitor `CollectTimesVisitor` that collects the discovery and finishing time of the visited vertices and also classifies the edges into tree, forward, backward and cross edges.

Note that in general there are multiple possibilities for DFS traversal, and therefore multiple possible times and edge classifications. The outcome depends on the order in which vertices and outgoing edges are processed. The provided graph implementation guarantees that the order follows the order in which the vertices/edges are created.
Use the unweighted graph from task 1 to write a unit test that verifies your visitor. This should yield the times and edge classification from the DFS example presented in the lecture.