Computer Science

CompSci 101 S1 C
– ASSIGNMENT THREE –

Assessment

Due: 3.00pm Monday, 7th May, 2007
Worth 4% of your final mark
There is no Peer review for this assignment.
The resources for this assignment can be found at the following URL:

www.cs.auckland.ac.nz/compsci101s1c/assignments/

Aim of the assignment

The goals of this assignment are to get you to:

- Practise translating your mental model of a system into a set of classes.
- Declare object variables, create and manipulate objects, and to pass and return objects to and from methods.
- Declare array variables, create and manipulate arrays of objects.

For this assignment you need to submit EIGHT Java source files:

- `A3Application.java`
- `A3Program.java`
- `Graph.java`
- `EdgeSet.java`
- `Edge.java`
- `NodeSet.java`
- `Node.java`
- `Keyboard.java`

Things to note:

You have been provided with the application, `A3Application.java`, the program file, `A3Program.java`, and with a version of `Keyboard.java`

Your code MUST be broken down into methods. Each method should primarily perform one task which is indicated by the method name.

Directed Graphs

A directed graph contains a set of nodes and a set of directed edges. A node is a primitive entity and has an id. A directed edge is an ordered pair of nodes, where one is the source node and the other is the destination node. The nodes mentioned in the graph's set of edges are all elements of the graph's set of nodes. *For this assignment, you may assume that all of the graph's nodes are involved in at least one of the graph's edges.*

You will need to develop classes that correspond to the above description, i.e., you will need to develop a class for each of the following: **graph**, **set of edges**, **edge**, **set of nodes**, **node.** The data type for node ids is **String.** Arrays must be used to store the elements of sets.

Program Functional Specification

Your program reads in a description of the graph in one format and writes out that graph in "dot" notation. "dot" is a language for describing graphs and a number of different viewers have been written to display graphs written in "dot". The "dot" language manual will be available on the assignment resource web page. Links to downloadable viewers can be found via the following link http://www.graphviz.org/About.php **graphviz** is one of the viewers available and can be used to view the output of your program.

Input Format: Your program reads in a directed graph (*digraph*) in the following format. All blank lines are ignored. The graph input consists solely of pairs of non-blank lines representing directed edges. Each line should have exactly one node id. The first line of each pair contains the source node id and the second line contains the destination node id. If the node id is more than one word it should be enclosed in double quotes (").

Output Format: Your program writes out the directed graph in the following format:

1. Preamble.
2. Edges.
3. Postscript.

The preamble consists of the following string "***digraph G {"***. The postscript consists of the string "*}*", a line stating the number of nodes, and a line stating the number of edges in the graph. Between the preamble and the postscript are lines describing the directed edges in the graph. Edges are specifed in the following format, "*source node -> destination node;"*. The edges are grouped by their source nodes, i.e., all the edges that have the same source node are specified together.

Example:

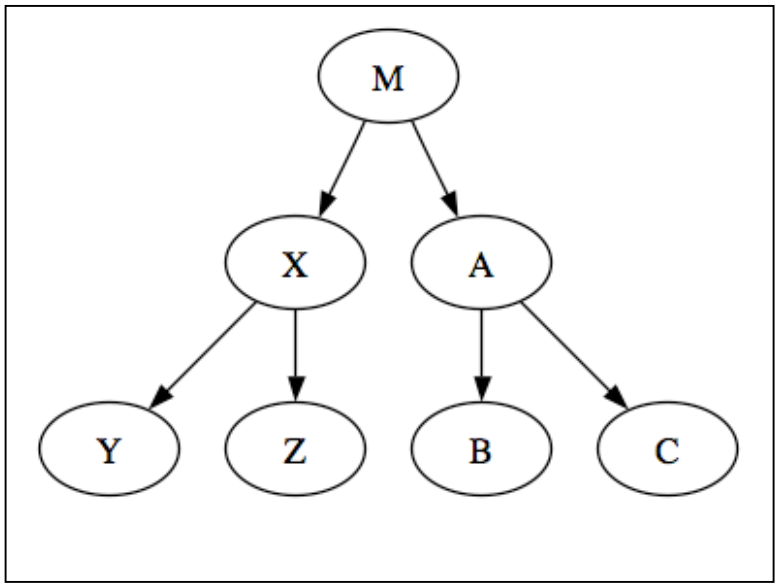
Input:

```
M
X
M
X
X
Y

A

B
X
Z
A
C
M
A
```

Output:

```
digraph G {
M -> X;
M -> A;
X -> Y;
X -> Z;
A -> B;
A -> C;
}
# nodes = 7
# edges = 6
```

Graph displayed by vizgraph using Hierarchical Layout

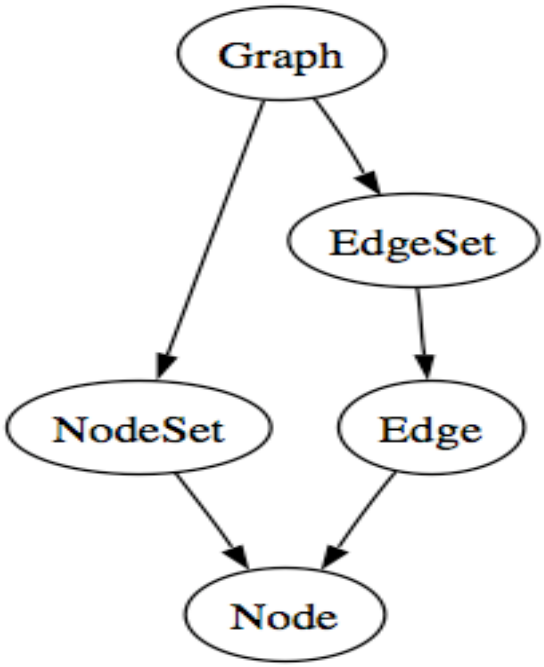


Notes:

1. Regardless of how many times a node id appears in the input, it always represents the same node (e.g., in this example there are only 7 nodes: **M X A Y Z B C**).
2. Regardless of how many times an edge appears in the input, it always represents the same edge (e.g., in this example there are only 6 edges, even though there are 7 pairs of lines with nodes, i.e., the edge **M -> X** appears twice in the input).
3. Blank lines are allowed to appear anywhere in the input.
4. In the output, all edges having the same source node are grouped together. For example, in the input there are two edges with **M** as the source node, one that points to **X** and one that points to **A**. These two edges appear as the first and as the last pairs of lines respectively. However, both edges appear grouped together in the output.

Program Design

You have been provided with the full **A3Program** class. You are to write the ***Graph*** library classes. Your implementation of the ***Graph*** library must have the five classes: **Graph, EdgeSet, Edge, NodeSet,** and **Node**. A **Graph** object will have at least an instance variable of type **NodeSet** and an instance variable of type **EdgeSet.** An **EdgeSet** object will have at least an **array** of **Edge** objects. A **NodeSet** object will have at least an **array** of **Node** objects. An **Edge** object will have two instance variables of type **Node**. The API's for these classes are shown in the appendix of this document.



Running your A3Program

In order to help you test your implementation of a ***Graph*** library, I have provided TWO input files: `test.txt` and `graphG.txt`. I have also provided a different version of the `Keyboard` class; this version allows you to use file redirection. In order to run your program you can run the application, `A3Application`, and enter data from the keyboard or you can run the application with the user input taken from either of the two input files, for example,

```
> java A3Application <test.txt
```

will execute the application using the user input from the file, `test.txt`, and produce the output shown above in "Program Specification". To produce the dot file for input to **vizgraph**, you would execute:

```
> java A3Application <test.txt > test.dot
```

You are expected to supply the input as a file via redirection as shown above. The input file should be an ascii text file created by your favorite text editor.

Submitting Files

You should submit the following files for this assignment through the web-based Assignment Dropbox:

```
A3Application.java
A3Program.java
Graph.java
EdgeSet.java
Edge.java
NodeSet.java
Node.java
Keyboard.java
```

MAKING MORE THAN ONE SUBMISSION

You can make more than one submission - every submission that you make *replaces* your previous submission. **Only your very latest submission will be marked.**

DO NOT SUBMIT SOMEONE ELSE'S WORK:

- If you submit an assignment you are claiming that you did the work. Do not submit work done by others.
- Do not *under any circumstances* copy anyone else's work – this will be penalized heavily.
- Do not *under any circumstances* give a copy of your work to someone else.
- The Computer Science department uses copy detection tools on the files you submit. If you copy from someone else, or allow someone else to copy from you, this copying will be detected and disciplinary action will be taken.

Marking Schedule

Style			
Comment at the top of each class.	**(3)**		
Good indentation.	**(3)**		
Good identifier names.	**(3)**		
Symbolic Constants used where appropriate.	**(3)**		
All instance variables are private.	**(3)**		/15

Normal Processing			
Handling of blank lines:			
Blank lines separating edges.	**(5)**		
Blank lines separating nodes within edges.	**(5)**		
Total input all blank	**(10)**		
Correct counting of edges.	**(10)**		
Correct counting of nodes.	**(10)**		
Correct stating of output preamble.	**(10)**		
Correct stating of output edges.			

All outgoing edges for a node together **(10)** All nodes have their outgoing edges **(5)** Edges only appear once **(10)** Correct stating of output postscript. **(10)**		/85

Grand Total		**/100**

Appendix – Class API's

public Graph API

constructors:

public Graph()

mutators:

public void addNode(Node) ensures that Node is in Graph's NodeSet by adding the Node if it is not already in the NodeSet

public void addEdge(Edge) ensures that Edge is in Graph's EdgeSet and ensures that the Edge's nodes are in the Graph's NodeSet

accessors:

public NodeSet getNodeSet()

public EdgeSet getEdgeSet()

public int getNodeCount() returns the number of nodes in the graph

public int getEdgeCount() returns the number of edges in the graph

public EdgeSet getOutgoingEdges(Node sourceNode) returns an EdgeSet containing the node's outgoing edges

public class EdgeSet API:

constructor:

public EdgeSet() the EdgeSet object is able to contain up to 40 Edge objects

accessors:

public Edge getEdge(int index) returns the edge associated with "index"

public String toString()

public int length() returns the number of edges that are in this EdgeSet

mutator:

public void addEdge(Edge edge) : adds edge to EdgeSet if not already present

public class Edge API:

constructor:

public Edge(Node sourceNode, Node targetNode)

accessors:

public Node getSourceNode()
public Node getTargetNode()
public String toString()
public boolean equals(Edge other)

public class NodeSet API:

constructor:

public NodeSet() is able to contain up to 20 Node objects

accessors:

public Node getNode(int index): returns the node associated with "index"
public int length( ) returns the number of nodes that are in this NodeSet

mutators:

public void addNode(Node node)

public class Node API:

constructor:

public Node(String name)

accessors:

public String getName()
public String toString()
public boolean equals(Node other)