CSC 207 Software Design
Winter 2015 — Exercise 2

1 Logistics

- **Due date:** 9:00pm Tuesday 03 March 2015
- **Group size:** Individual
- **Topics:** Java Generics, Exceptions

For the rules and procedures for the exercises, including how to submit, please see the Exercises page of the course website.

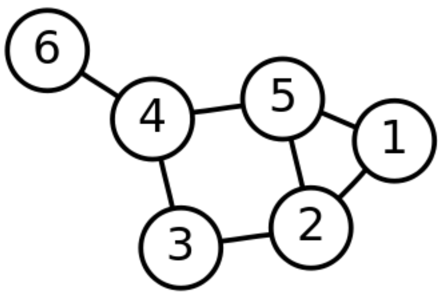
2 What to do for this exercise

1. Your individual `svn` repository now contains a new directory called `E2`. It contains the starter code for this exercise. Checkout and study the starter code.
2. Complete/implement Java classes `Graph.java` and `NoSuchNodeException.java`, so that obey the specifications below and the descriptions in the starter code.
3. To submit your work, add and commit your changes to your repository.

   Do **not** commit the files and directories generated by Eclipse, such as `bin`, `doc`, `.project`, etc. Marks will be deducted if you submit these.

3 Abstract Data Type: Graph

In this exercise, you will implement a data structure to represent a graph. A graph consists of nodes and edges. An edge is a connection between two nodes. For example, here is a graph with 6 nodes (nodes 1, 2, 3, 4, 5 and 6) and 7 edges (between nodes 1 and 2, 1 and 5, 2 and 5, 2 and 3, 5 and 4, 3 and 4, and 4 and 6).



Here is more information about the specific type of graph that we'll work with:

- Each node in the graph has a unique identifier.
- Edges are undirected. To represent this, if there is an edge from node X to node Y, there will also be an edge from node Y to node X.
- If an edge already exists between two nodes, then no additional edges can be added between them.
- A node cannot have an edge connecting it to itself (i.e. no self edges are allowed).

4 Specifications for `Graph.java`

Your task is to complete the implementation the class `Graph`. To implement `Graph` we will use a `Map`, where each key is a `Node` and each value is a `Set` of `Node`s adjacent to it. Two nodes are adjacent if there is a (single) edge connecting them. For example, in the graph above, node 4 is adjacent to nodes 3, 5, and 6.

The `Graph` class must be *generic*, so that we can create `Graph`s of `Node`s that store `Integer`s, `String`s, and any other objects as values. Implement the missing methods in the starter code.

5 Specifications for `NoSuchNodeException.java`

An exception class `NoSuchNodeException` must be a **checked exception**, and needs only two members: a no-argument constructor, and a one-argument constructor that takes a `String` message. Each constructor should simply call the corresponding constructor of the parent class.

6 Marking

The mark for correctly named files that compile and run, producing the correct output, is 3 marks.

If you submit files with the correct names, but they are not in the correct directory, or do not compile, or do not belong to the correct package, or do not run, or do not produce the correct output, **or does not pass all tests**, the solution will receive 0 marks.

7 Checklist

Have you...

- tested your code on the lab computers using **Java 1.7**?
- committed the correct files in the correct directory?
- verified that your changes were committed using `svn list` and `svn status`?
- checked the pre-marking results, made any necessary changes, and re-committed if necessary?