CSC207/B07 Software Design
Fall 2013 — Exercise 2

1 Logistics

- **Due date:** 10:00pm Thursday 31 October 2013
- **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.

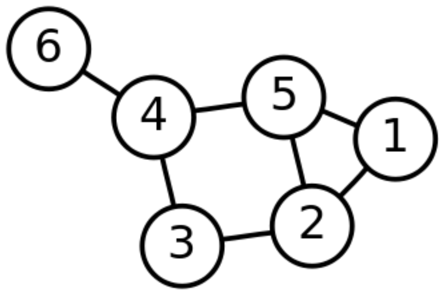
2 What to do for this exercise

1. Write Java classes `Graph.java` and `EdgeException.java` that obey the specifications below. They must be in project `E2soln`, in package `e2soln`.
2. To submit your work, commit `E2soln`, `src`, `e2soln`, and your Java files under the existing directory `E2`.

   Do **not** commit the files and directories generated by Eclipse, such as `bin`, `.project`, etc.

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 is unique.
- Each node in the graph is connected to at least one other node in the graph.
- Edges are undirected. To represent this, if there is an edge from node 1 to node 2, there will also be an edge from node 2 to node 1.
- 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 implement the class `Graph`. To implement `Graph` use a `Map`, where each key is a node and each value is a `Set` of nodes 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 `Integer`s, `String`s, and any other objects. Note that you do **not** need to define a class for nodes. In our (simplistic) implementation, a graph of `Integer`s has `Integer` objects as keys in the `Map`, a graph of `String`s has `String` objects as keys, etc. Implement these methods:

- A constructor with no parameters.
- A method `addEdge` that takes two nodes and adds an edge between them, if no such edge currently exists. Both directions of the edge should be added to the `Map`. If one of the nodes does not exist, it creates it. Assume that the two input nodes are distinct. If the graph is non-empty, assume that at least one of the two input nodes is in the graph.
- A method `areAdjacent` that takes two nodes and returns `true` iff there is an edge between them in the graph. If one or both of the given nodes is not in the graph, it throws an `EdgeException` (see its specification below).
- A method `getNodes` with no parameters that returns the nodes in the graph as a `Set`.
- A method `getNeighbours` that takes one node and returns a `Set` of that node's neighbours.
- For testing purposes, add the three methods from `starter.txt` to your `Graph` class. Note that the starter code uses `T` as the type variable. You may need to change this variable name to match your solution code.

Note: besides method `areAdjacent`, none of the specified methods should throw an exception.

5 Specifications for `EdgeException.java`

An exception class `EdgeException` must be a **checked exception**, and needs only two members: a no-argument constructor, and a one-argument constructor that takes a `String`. 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, 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?