1. Announcements:
   (a) Tomorrow is last day to sign up for early sitting of final, 12/14 9:30am.
   (b) Darwin contest! Tomorrow night, 7:30, Physics 203. Prizes for all!
   (c) Submit (working) creatures by midnight, tonight, in Creatures/Creature.txt.

2. Graphs.
   (a) Terminology: node or vertex (not “vertex”), edge, directedness, path, cycle, and degree.
   (b) Interface:
      i. Graph<V,E> has vertex labels of type V, edge labels of type E.
      ii. Acts like a typical Structure of elements of type V, but also supports relations between vertices:
         addEdge(u,v,e),
         E removeEdge(u,v),
         Edge<V,E> getEdge(u,v), and
         containsEdge(u,v).
      iii. Note important difference between edge labels (can be null) and Edge<U,V> class:
         A. The Edge<V,E> class has here, there and label methods.
         B. On undirected edges here and there are arbitrarily ordered, but consistent.
         C. On directed edges, the edge goes from here to there.
      iv. “Visitation” methods:
         visit(v),
         isVisited(v),
         visitEdge(Edge<U,V> e),
         isVisitedEdge(Edge<U,V> e).
      v. size(), edgeCount(), and degree(v).
      vi. Iteration: iterator() (over vertices), edges() (over edges), and for vertices, neighbors(), over reachable neighboring nodes.

(c) Two common implementations:
   i. Adjacency matrix. Edges are stored in a matrix at (u, v) if the edge goes from u to v. This is only efficient if the number of edges approaches $n^2$ (for an n node graph). Undirected graphs lead to symmetric matrices.
   ii. Adjacency list. Edges are stored in lists of entries in a Map (a Hashtable is commonly used). This implementation is preferred if the graph is sparse ($O(n)$ edges). Undirected graphs take twice the adjacency list space unless you wish to take more time.

(d) Engineering approach:
   i. Graph is the general graph interface.
   ii. GraphMatrix and GraphList are abstract base classes for graphs of each type.
      A. These classes support directed or undirected graphs of each type.
      B. Most methods are (at least partially) implemented, but not those that involve user-observable manipulation of edges.
      C. Directed and undirected variants of each type of graph are extensions of the abstract base class.
   iii. GraphMatrix keeps the association (GraphMatrixVertex) of nodes and row indices in a dictionary. Unassigned row indices are kept in a free list.
   iv. GraphList keeps a similar association, but instead of an index, each vertex keeps a singly linked list of neighbors. The GraphListVertex class has many methods for maintaining the neighbor list.
3. Algorithms.

(a) Depth-first search from a node: visit any unvisited node and then recursively visit neighbors.

(b) An approach to connected components (undirected): depth-first search from a source.

(c) Transitive closure–add an edge between any nodes that have a path between them (Warshall’s algorithm).

(d) All-pairs minimum distance–compute new edges or update old edges with a minimum distance (Floyd’s algorithm).

(e) Minimum spanning tree: visit edges that bring the closest unvisited node into the graph. Start anywhere.

(f) Single-source shortest path. Start at a specific location and add edges with minimum total distance to unvisited nodes. (Dijkstra’s algorithm).