TP3

DE52

stephane.galland@utbm.fr

Implementation of a graph

00 | Objectif

The objective of this lab session is to create an information graph structure by applying implementation principles derived from software engineering.

The configuration of Eclipse should be the same as for previous sessions.

This lab assignment will be completed over

TWO OR THREE SESSIONS

01 | Exercises

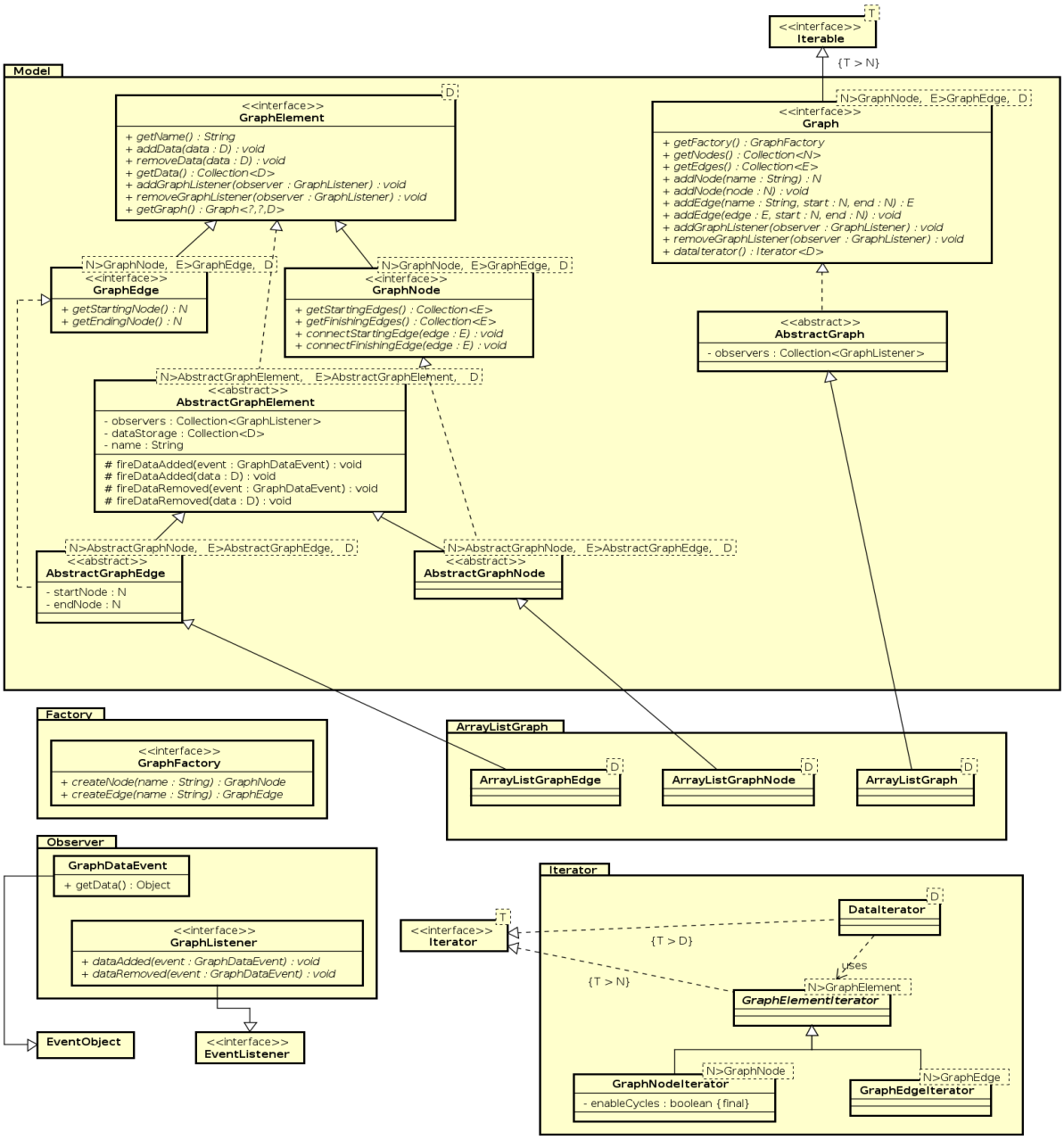
01.1 Implementation of a graph

A graph is a data structure consisting of nodes (nodes) and edges (edges). The edges connect the nodes. An edge can only connect two nodes (or a single node if both ends of the edge are connected to that node).

In the context of this lab, we will only consider directed graphs, meaning their edges have a single direction (from the start node to the destination node) and can only be traversed in that direction.

A graph is a structure that can be used for storing particular information. However, for this lab, we ask you to allow the storage of data of a predefined type (T) in the nodes or edges.

The UML class diagram below gives you the structure and relationships between the types to be implemented at the end of the lab.



You must adhere to the basic principles of programming inspired by software engineering best practices, namely:

1. Use clear names in English, as your code must be readable by any developer, regardless of their native language.

2. Correct the code so that there are no errors or warnings, as each error or warning indicates an anomaly related to good programming practices.

3. Write clear documentation in English (you will have warnings to help you).

4. Apply design patterns as much as possible: https://en.wikipedia.org/wiki/Design_Pattern

Work to be done:

A) Generic Graph Structure

- Write the Graph, GraphNode, and GraphEdge interfaces representing a graph and its components. These types must take 3 generic parameters:
  1. N: represents the type of nodes
  2. E: represents the type of edges
  3. D: represents the type of data stored in the nodes and/or edges
- Write the abstract classes corresponding to each of the three interfaces implemented above.
- Write a concrete implementation for each of the three abstract classes using Java arrays.

B) Design Pattern: Factory

- The 'Factory' design pattern allows the creation of objects (new operator) to be externalized in a specific class called Factory.
- Define the GraphFactory interface.
- Create an implementation of this GraphFactory in ArrayListGraph and use it to create the nodes and edges.

C) Design Pattern: Observer

- Write the GraphDataEvent interface as an object containing information corresponding to a modification of the data in the graph.

- Write the GraphListener interface representing an observer on a graph.
- Integrate the 'Observer' design pattern into the graph structure. Observers can observe a node, an edge, or the entire graph.

E) Design Pattern: Iterator

- Two implementations of the 'Iterator' design pattern are necessary: one for iterating over the nodes of the graph, and a second for iterating over the data stored in the graph.
  - Note: The graph can be cyclic, meaning it contains cycles in the paths that can be traversed from node to node. Therefore, infinite loops must be avoided during iterations.
- Create the iterator for the nodes of a graph.
- Create the iterator for the data stored in a graph.

G) Test Program

Write a 'main' function that performs the following actions:

1. Create a graph of integers.
2. Add random integers to the nodes and edges.
3. Display the nodes of the graph.
4. Display the integers stored in the graph.

AT THE END OF THE TWO/THREE LAB SESSIONS, SEND A ZIP FILE CONTAINING THE JAVA CODE YOU PRODUCED DURING THE LAB