Luleå University of Technology
Computer Science
Håkan Jonsson

January 18, 2017
D0010E

# Lab 3

## – Breadth-First Graph Traversal –

## 1 Introduction

In this assignment you are asked to write two classes that will be part of a much larger program. You have to do this in isolation without any knowledge of the full source code of the large program. Rather, you must rely only on UML-diagrams and general ADT specifications that describe the functionality your classes must provide.

You are expected to work on your own. You may present your work for grading as soon as you are done with it all and have tested it properly. Section 3 contains details on how the grading is done.

## 2 Background story and assignment

As part of a large development team at *Acme Programming, Inc.* it is your task to implement two Java classes for use in a program that performs and visualizes breadth-first traversals of graphs. Convinced that the program will be a smash-hit for the company (that is, his huge pile of stock options), your boss has already handed over the necessary documents to the team. Among them is a UML diagram that describes the complete program (Section 5). You read it to get a feeling for how your parts fit into the large program. At first, you are baffled by the many classes and interfaces this diagram contains but soon you identify the two classes you are responsible for.

Your team members have already written their parts of the program and now eagerly await yours over a beer at the local pub. To enable you to test your classes, and run the whole program, in their absence they have put together two JAR files (where JAR stands for ***J**ava* ***AR**chive*) together with an example gridgraph file, and made them available from the web page containing the lab instruction. The first JAR file, `FIFOmain.jar`, is a test program to verify that your `FIFO` queue you are to write in Task 1 below works the way it should[1]. The second, `BFTmain.jar`, implements a *Breadth-First Traversal* of a graph. The code in this JAR, together with your two classes, make up the complete program.

**Task 1.** Create a new Java project called `Lab3` in Eclipse, download the two jar files from the web and put them in the directory of the project. Then, add them to the build path of your project. Add them as jar files and not as so-called *external* jar files.

Having setup your project in Eclipse you can move on to the actual main task.

**Task 2.** According to your instructions, your first class should be an implementation of the interface `Queue`. You find it specified in a UML diagram and ADT specification (Section 6) your boss left on your desk just before he left for what now, several days later, seems like an unusually long round of golf.

- Your implementation should realize a *First-In-First-Out* (FIFO) queue. In this kind of container, the element that was added earliest, among all elements currently in the queue, should be the next element returned by method `first` and removed by method `removeFirst`.

[1]The test program does some simple operations using the `FIFO` class you have written to check for common errors. However, it is not guaranteed to find all possible errors. Therefore, passing the test does not necessarily guarantee that your implementation is correct.

- Your class must have the name `FIFO`, or it will not work together with the other classes provided by your group members.
- Tip: You can use some collection class from Java's standard library (for example, `ArrayList`) to store elements of `FIFO`.

The queue is one vital part and the other is the part of the program that reads input from a file.

**Task 3.** Your second task is to implement the class `GraphIO`, which is described in a separate UML diagram and ADT specification (Section 7).

- This class reads a specification of a graph from a text file (a gridgraph) and adds nodes and edges to a graph internal to the program according to the specification.
- This class must have the name `GraphIO`, or it will not work together with the other classes provided by your group members.

The complete program should open a window displaying a graph with one of the nodes, chosen at random, colored gray. If you click in the window the node should become black while its neighbors turn gray. Repeat clicking until all nodes are black and try to understand in what order the nodes are colored. What is shown below the graph?

More information on how the graph is traversed can be found on Wikipedia:

`http://en.wikipedia.org/wiki/Breadth-first_search`.

## 3 Examination

Presenting your work on this assignment for grading means you have to:

1. Demonstrate that your class `FIFO` works with the test program `FIFOmain`. Main class is `FIFOmain`.
2. Demonstrate that your class `GraphIO` can read the gridgraph file provided and visualize the traversal of the graph. Here, the main class is `BFTmain` and the name of the gridgraph should be specified as an argument to the program.
3. Likewise, demonstrate that it works also for your own gridgraph, that must include at least 8 nodes and 12 connections.
4. Reason and argue that each method of your class `Queue` fully meets the ADT specification and is programmed in a sensible way.
5. Answer various questions regarding your programs.

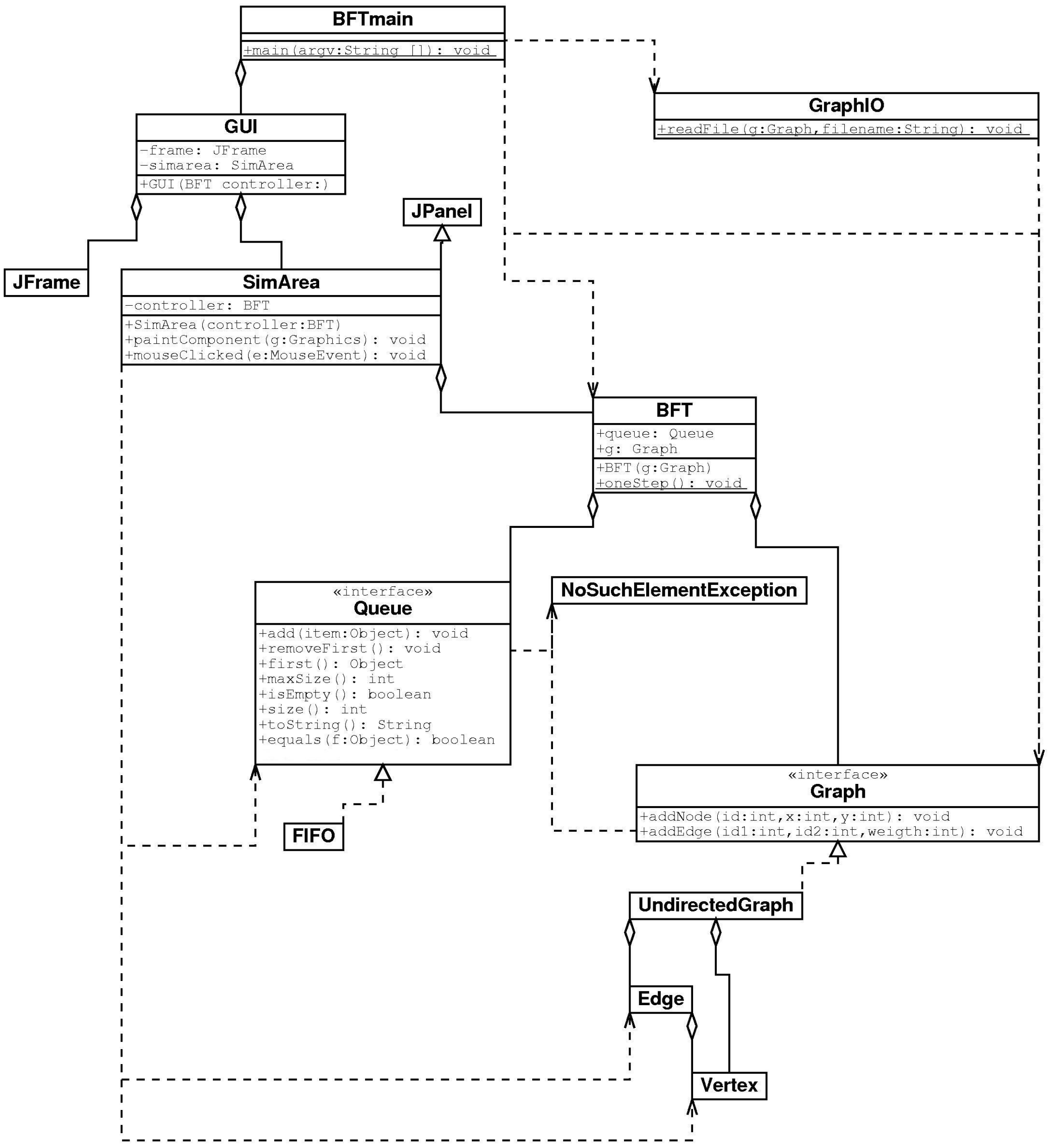
## 4 Notes and restrictions

- To store elements in your `FIFO` class, you can use an array or any class from the Java standard library that inherits from `AbstractList`. However, you may not use classes that implement the interface `Queue` from the Java standard library.
- Two `FIFO` queues are equal if, and only if, they contain the same number of elements and the same element when compared in the same order. The method `equals` in the `FIFO` class should work like this.
- If you use a class from the Java standard library to store elements (such as`ArrayList`), you may not use the method `equals` provided by that class. You must write your own method.
- The method `toString` of the `FIFO` class should use the `toString` methods of the individual elements of the queue.
- When we say that a method throws a certain exception, this normally means that it is not allowed to throw any other exception.

## 5 The complete program: UML

In this UML diagram

- a line with a triangle denotes an 'is a' relation
- a line with a diamond denotes a 'has a' relation
- a dashed line with an arrow denotes a 'knows of' relation
- a dashed line with a triangle denotes an 'implements' relation



## 6 Queue: UML & ADT Specification

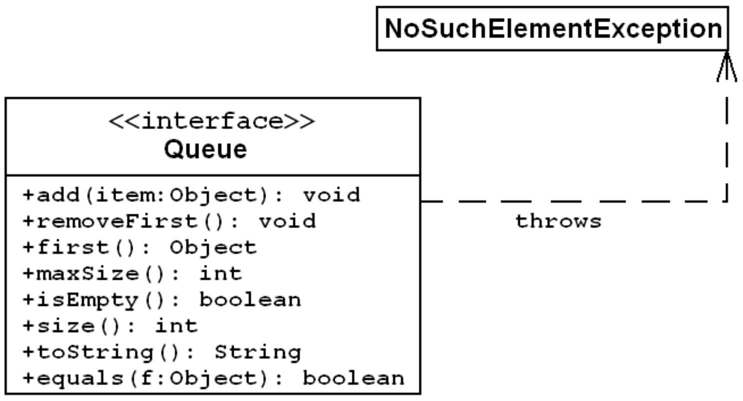


Figure 1: UML-diagram of `Queue`.

### Queue ADT specification

#### Domain

Every `Queue` consists of a sequence of elements. An element is a reference of type `Object`. References might be `null`.

#### Query Methods

`public int size()`
This method returns the number of elements in this queue.

`public int maxSize()`
This method returns the maximum size this queue has had since it was created.

`public boolean isEmpty()`
This method returns `true` if, and only if, the size of this queue is 0. Otherwise, `false` is returned.

`public Object first()`
This method returns the first element in the queue. If the queue is empty, a `NoSuchElementException` is thrown.

`public boolean equals(Object f)`
This method throws a `ClassCastException` if `f` is not of the same type as this class. This method returns `true` if, and only if, all the following holds:

- `this` and `f` have the same size.
- For every position $i$ in this queue:
  - If the element at position $i$ is `null`, the corresponding element at position $i$ in `f` also is `null`.
  - If the element at position $i$ instead is a reference to an object `obj1`, the corresponding element at position $i$ in `f` is also a reference to an object `obj2`, and `obj1.equals(obj2)` is `true`.

Otherwise, this method returns `false`. In particular, it does *not* throw any exception, such as a `ClassCastException`.

`public String toString()`
This method returns a string beginning with `"Queue: "` followed by the following, for each element *elem* in the queue:

```
"(" + String.valueOf(elem) + ") "
```

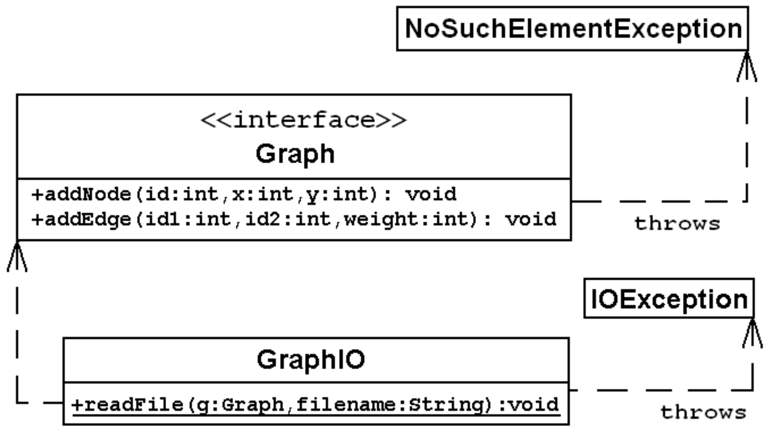
Note that the elements must be listed in order. Note that definition forces the string to end with a whitespace.

### Update Methods

`public void add(Object item)`
This method adds the object `item` to the end of the queue.

`public void removeFirst()`
This method removes the first object from the queue. If the queue is empty, a `NoSuchElementException` is thrown.

## 7 GraphIO: UML & ADT Specification


NoSuchElementException
<<interface>>
Graph
+addNode(id:int,x:int,y:int): void
+addEdge(id1:int,id2:int,weight:int): void
throws
IOException
GraphIO
+readFile(g:Graph,filename:String):void
throws

### GraphIO ADT specification

`static public void readFile(Graph g, String filename)`

**Pre-condition:** The string `filename` contains the name of a file containing a description of a graph that can be opened and read. If not, an `IOException` is thrown.

**Post-condition:** Nodes and edges according to the description in the file `filename` has been added to the graph `g`.

The file `filename` with the description is line-based. If the contents does not follow the description given below, the program might throw other exceptions but it is only when the pre-condition is violated a *checked*[2] an exception like `IOException` might be thrown.

The first line of the file contains an integer, $n$, stating how many nodes there are in the graph. The following $n$ lines contains three integers each. On each line, these is one space between the first two integers and one space between the last two integers. The integers represents the identity, the x-coordinate, and the y-coordinate of the node (in this order).

The rest of the file also contains lines with three integers, but these represents edges between edges in the graph. The first two integers on a line are identities of two connected nodes and the third (and last) integer is the weight of the edge. See the contents of the provided gridgraph for an example.

Tip: Use a `Scanner` object to read and interpret the file.

[2] A method that might throw a checked exception must catch the exception, using a `try-catch` statement, or be declared with a `throws` clause.