Lab 4: Representation of graphs

Task 1: *Application*

a) Create package *worksheet4*

b) Create a classe *GraphPainter* und copy the contents of file GraphPainter.java (provided by moodle).

c) Class *GraphPainter* contains a main method with test program.

- Among others, you find an adjacency matrix here. Its variation enables you to test the behaviour of your program.
- If you wish to add vertices, the paint method needs to be extended accordingly. In particular, the *layout* has to be adjusted when having more vertices.

d) An iterator for the class *AdjacencyList* is to be provided. You have to verify in the test program whether the iterator works.

Task 2: *Create a class for the visualisation(call it Visualisation) and connect it to the GraphPainter-class, in order to have a more modular system.*

Task 3: *Write a program which represents a* graph*, that is to be implemented by both an adjacency list and an adjacency graph.*

a) Provide a class graph Graph which can employ both adjacency structures.

b) Implement everything regarding the UML diagram on the last page (Abbildung. 1).

c) For the sake of simplicity it is sufficient to store integer values in the graph. They should match the values of the vertices shown on the display (in the main test program).

d) In the class *Graph* a method *somePath(int u, int length)* is to be provided, which follows an arbitrary path from vertex *u* which has the provided *length*. Invent a simple algorithm for generating such a path. But be careful to avoid cycles. The resulting path is to be stored in an object of class AdjacencyList.

Each group has to define an own path and implementation of *somePath(int u, int length)*. This method is the main challenge of this last exercise.

You call method *somePath(int u, int length)* from your test program *main*. The specified path is to be painted in red. For this purpose, the method *setAPath(aPath)* of *GraphPainter* is to be called.

There is a hidden bug in the GraphPainter. Who is able to find it?

General information:

a) Use the programming language Java.

b) Denote the Package *worksheetk*, e.g. *worksheet*1.

c) Use the following conventions in your programs:

- *english* names
- meaningful, but short names
- camelCase
- classes, interfaces, static variables start uppercase
- variables start lowercase
- constants are completely uppercase

d) further code conventions

- in general, declare everything as to be private; only where necessary, protected or public can be used for methods (attributes are always private, and only in a few cases protected)
- curly braces start at the end of a row
- closing curly braces are to be written on a new row (after carriage return line feed)
- avoid unnecessary empty rows
- empty rows are to be used after package declaration, import statements, before classes, interfaces, and methods
- Order of variables and methods within classes and interfaces:
  1. constants and variables (both called attributes)
  2. constructors
  3. getter and setter for attributes (only those which are necessary!)
  4. all other public methods
  5. further (protected and private) methods

e) Each class and each interface is in a *separate* file.
(Exceptions are inline classes)

f) At the top of each file you write: *authors, date, purpose of the class* and which *public-methods* exist for which purpose.

g) Code and PDF documents are to be submitted timely to Emil.

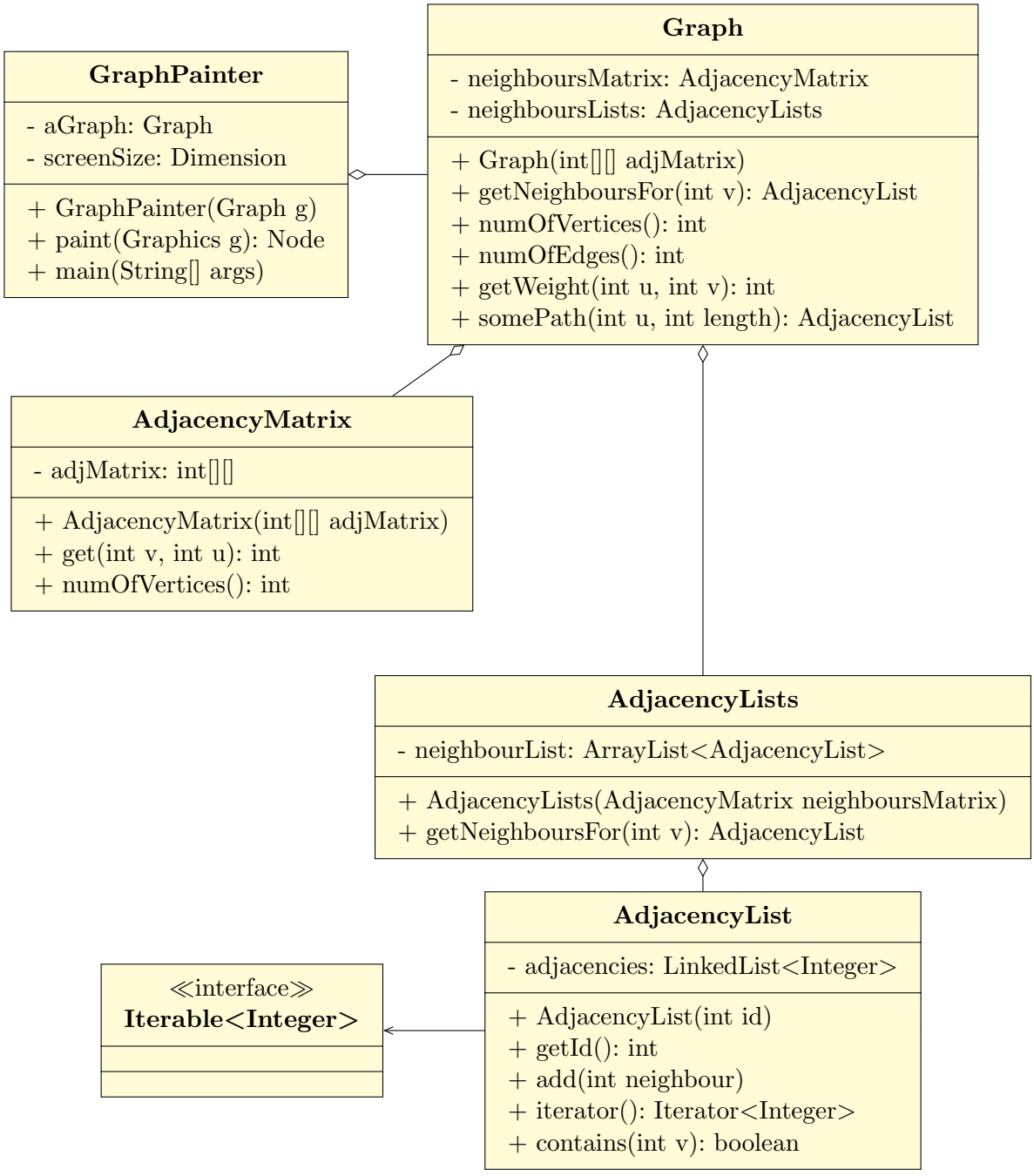

Abbildung 1: UML diagram for the classes to be implemented.