# Assignment 4

SFWR ENG 2AA4

Files due Mar 11, E-mail partner due Mar 12, Lab report due Mar 16

The purpose of this software design exercise is to create, use and test a Java program that stores a map of a hallway network, together with the location of eggs, blockages and openings.

## Step 1

Write a module that creates a point ADT. It should consist of a Java code file named `PointT.java`. The specification for this module is given at the end of the assignment.

## Step 2

Write a module that creates a node ADT. It should consist of an Java file named `NodeT.java`. The new module should follow the specification given at the end of the assignment.

## Step 3

Write a module that creates an edge ADT. It should consist of an Java file named `EdgeT.java`. The new module should follow the specification given at the end of the assignment.

## Step 4

Write a module that implements an abstract object (not an abstract data type) to store the map. It should consist of Java files named `Map.java`. The new module should follow the specification given at the end of the assignment. Although efficient use of computing resources is always a good goal, your implementation will be judged on correctness and not on performance.

## Step 5

Write a module that uses JUnit to tests all of the other modules together. It should be an Java file named `AllTests.java` that tests all of the other modules. `AllTests.java` should use unit testing files to test the other classes. That is, there should be a separate file to test each of the other modules. The names of these JUnit testing files should be as follows: `TestPointT.java`, `TestNodeT.java`, `TestEdgeT.java`, and `TestMap.java`. You do not need to write a makefile for this assignment.

The test sets should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your lab report.) Please make an effort to test normal cases, boundary cases, and exception cases. Your grade will depend in part on the correctness of your program, which can in part be improved by careful consideration of the test cases.

## Step 6

Submit the files `PointT.java`, `NodeT.java`, `EdgeT.java`, `Map.java`,`TestPointT.java`, `TestNodeT.java`, `TestEdgeT.java`, `TestMap.java` and `AllTests.java` using subversion. In addition submit any Java code that you need to write for the exceptions required by the specification. This must be completed no later than midnight of the deadline for file submission.

E-mail the `EdgeT.java` file to your assigned partner. (Partner assignments will be posted on WebCT, on the day after the initial submission.) Your partner will likewise e-mail you his or her files. These e-mails should be traded by midnight of the day following the file submission.

## Step 7

After you have received your partner's files, replace your corresponding files with your partner's. Do not make any modifications to any of the code. Run your test module and record the results. Your evaluation for this step does not depend on the quality of your partner's code, but only on your discussion of the testing results.

## Step 8

Write a report that includes the following:

1. Your userid on the first page.
2. Your name and student number.
3. Your partner's `EdgeT.java` file.
4. The results of testing your files (along with the rational for test case selection).
5. The results of testing your files combined with your partner's files.
6. A discussion of the test results and what you learned doing the exercise. List any problems you found with (a) your program, (b) your partner's module, and (c) the specification of the modules.
7. A comparison between Java and OCaml.
8. A copy of the part of your log book relevant to this lab exercise.

A physical copy of the lab report is due at the beginning of the lecture on the assigned due date.

### Notes

1. Place all submitted files in your svn repository in the folder `Assig4`.
2. Please put your name and student number at the top of each of your source files. (You should remove the student number before e-mailing any files to your partner.)
3. Your program must work in the ITB labs on moore when compiled by javac.
4. If your partner fails to provide you with a copy of his or her files by the deadline, please tell the instructor via e-mail as soon as possible.
5. If you do not send your files to your partner by the deadline, you will be assessed a **10% penalty** to your assignment grade.
6. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**
7. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes.

# Point ADT Module

## Template Module

PointT

## Uses

N/A

## Syntax

### Exported Constants

TOLERANCE $= 1 \times 10^{-4}$

### Exported Types

PointT = ?

### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new PointT | real, real | PointT | |
| xcoord | | real | |
| ycoord | | real | |
| dist | PointT | real | |
| equal | PointT | boolean | |

## Semantics

### State Variables

$xc$: real
$yc$: real

### State Invariant

None

**Assumptions**

None

**Access Routine Semantics**

new PointT $(x, y)$:

- transition: $xc, yc := x, y$
- output: $out := self$
- exception: none

xcoord:

- output: $out := xc$
- exception: none

ycoord:

- output: $out := yc$
- exception: none

dist$(p)$:

- output: $out := \sqrt{(xc - p.\text{xcoord})^2 + (yc - p.\text{ycoord})^2}$
- exception: none

equal$(p)$:

- output: $out :=$ self.dist$(p) \leq$ TOLERANCE
- exception: none

## Node Module

### Template Module

NodeT **inherits** PointT

### Uses

PointT

### Syntax

#### Exported Types

NodeT = ?

nodeTypeT = { JUNCTION, EGG, BLOCKAGE, OPENING }

#### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new NodeT | real, real, nodeTypeT | NodeT | |
| ntype | | nodeTypeT | |

### Semantics

#### State Variables

*nt*: nodeTypeT

#### State Invariant

None

#### Assumptions

None

**Access Routine Semantics**

new NodeT $(x, y, n)$:

- transition: $xc, yc, nt := x, y, n$
- output: $out := self$
- exception: none

ntype():

- output: $out := nt$
- exception: none

# Edge Module

## Template Module

EdgeT

## Uses

PointT, NodeT

## Syntax

### Exported Types

EdgeT = ?

### Exported Constants

TOLERANCE = $1 \times 10^{-5}$

### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| new EdgeT | NodeT, NodeT | EdgeT | |
| node1 | | NodeT | |
| node2 | | NodeT | |
| length | | real | |
| is_horizontal | | boolean | |
| equal | EdgeT | boolean | |

## Semantics

### State Variables

$n1$: NodeT
$n2$: NodeT

### State Invariant

None

**Assumptions**

None

**Access Routine Semantics**

new EdgeT ($\mathit{nod1}, \mathit{nod2}$):

- transition: $n1, n2 := \mathit{nod1}, \mathit{nod2}$
- output: $\mathit{out} := \mathit{self}$
- exception: none

node1():

- output: $\mathit{out} := n1$
- exception: none

node2():

- output: $\mathit{out} := n2$
- exception: none

length():

- output: $\mathit{out} := n1.\mathrm{dist}(n2)$
- exception: none

is_horizontal():

- output: $\mathit{out} := |n1.\mathrm{ycoord}() - n2.\mathrm{ycoord}()| \leq \mathrm{TOLERANCE}$
- exception: none

equal($e$):

- output:

  $\mathit{out} := (n1.\mathrm{equal}(e.\mathrm{node1}) \wedge n2.\mathrm{equal}(e.\mathrm{node2})) \vee (n1.\mathrm{equal}(e.\mathrm{node2}) \wedge n2.\mathrm{equal}(e.\mathrm{node1}))$

- exception: none

## Map Module

### Module

Map

### Uses

EdgeT

### Syntax

#### Exported Access Programs

| **Routine name** | **In** | **Out** | **Exceptions** |
|---|---|---|---|
| init | | | |
| add | EdgeT | | ALREADY_IN_MAP |
| del | EdgeT | | NOT_IN_MAP |
| contains | EdgeT | boolean | |

### Semantics

#### State Variables

$s$: set of EdgeT

#### State Invariant

None

#### Assumptions

init() is called before any other access program.

#### Access Routine Semantics

init():

- transition: $s := \{\}$
- exception: none

add($e$):

- transition: $s := s \cup e$
- exception: $exc := \exists(f : \text{EdgeT} | f \in s \wedge f.\text{equal}(e)) \Rightarrow \text{ALREADY\_IN\_MAP}$

del($e$):

- transition: $s := s - e$
- exception: $exc := (\neg\exists(f : \text{EdgeT} | f \in s \wedge f.\text{equal}(e))) \Rightarrow \text{NOT\_IN\_MAP}$

contains($e$):

- output: $out := \exists(f : \text{EdgeT} | f \in s \wedge f.\text{equal}(e))$
- exception: none