Lab 3: Modelling

Aim

This lab class gives you an opportunity to:

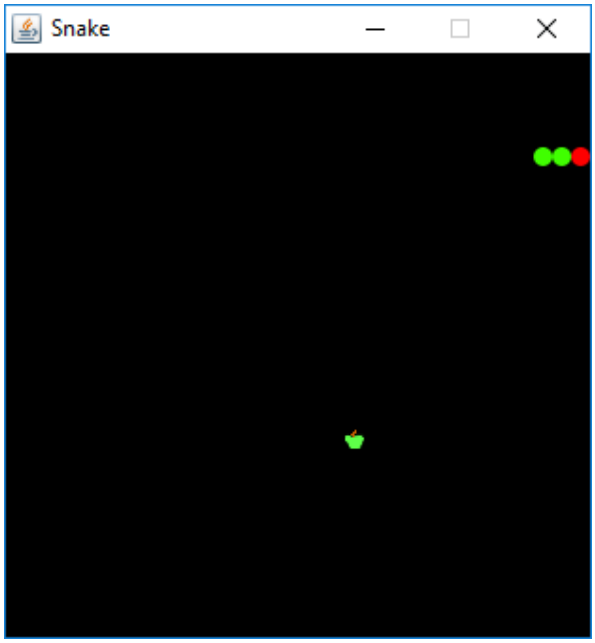
- create ADT class diagrams from a specification
- create a Java interface from a UML diagram;
- gain further experience with multiple ADTs which have been specified by an interface and implemented by a class; and
- practise your Java programming.

Context

Snake is an older classic video game. It was first created in late 1970s.

In this game the player controls a moving snake which has a red head and a green segmented body and is initially of length three and moving to the right. The objective is to move the snake around the board (up, down, left, right) and to eat as many apples as possible. Each time the snake eats an apple its body grows by one segment. The snake must avoid the edges of the board and its own body; the game ends if the head collides with either of these.

A sample implementation from *zetcode.com* (`http://zetcode.com/tutorials/javagamestutorial/snake/`) is below:



Approach

To implement this ourselves (and to use it as an opportunity to learn about ADTs) we'd need to model the game. If we look at the description and image above we can identify a number of 'things' (nouns) which will become our ADTs. We can also identify a number of 'doing words' (verbs) which will become the methods of our ADTs. Lastly we can identify a number of 'describing words' (adjectives) which will become the properties of our ADTs.

It is common to use getter/setter methods to allow an object's property to be obtained or adjusted. For a property x of type X, these would normally be declared:

```
public X getX();
public void setX(X newX);
```

Tasks

1. Using pen-and-paper, a UML editor such as *StarUML*, or a general word processor such as *Microsoft Word*, construct UML diagrams for the `Board`, `Snake`, `Segment`, and `Apple` ADTs. Don't forget to include getter and setter methods or the constructor.

   If you would like to experiment with the program, please download the finished product from *MyLO*, open the files in *DrJava*, compile and execute it!

2. If you finish the task early, convert your UML diagrams to Java interfaces, creating the files in *DrJava* as `BoardInterface.java`, `SnakeInterface.java`, `SegmentInterface.java`, and `AppleInterface.java` respectively.