# Data Structures and Algorithms

Assignment One

Due Friday 25 March 2016

Please ensure that relevant files are submitted in the correct assignment folder on AUT online by 11.59pm on the due date. Put your projects into a folder and zip it up with your name and ID. Only classes from the Java standard library may be used for this assignment. Please note that marks are allocated for program design and readability as well as correctness.

**Questions:**

1. The class `Boid` (available on AUTOnline) represents a single movable object that can be drawn and run as a thread. The boid can move around the bounds of a panel with specified velocities in each of its x and y directions. Each `Boid` needs to know the position and velocities of the nearby (neighbouring) boids around it. However the class is not thread safe. Make the necessary changes to make this class thread safe and avoid any concurrency problems.

   **(10 marks)**

2. Using the following UML class diagram create a data structure called `BoidFlock` that holds `Boid` objects in a list. The default constructor just creates an empty flock and initializes the `boidList` using an appropriate data structure, where the 2nd constructor creates a flock with the specified number of boids to start with. The `addBoidToFlock` method should add a new `Boid` to the `boidList` using random positions (0 - panel width and height) and velocities ( `Boid.MAXSPEED`), it also needs to pass itself as a reference to the newly created `Boid`, then should start the object up as a thread. The `removeBoidFromFlock` should remove an arbitrary `Boid` object from the list, also stopping its thread execution, `destroyAllBoids` should stop all boid threads running and clear the list, `getNumberOfBoids` returns the current number of boids in the list. The `getNeighbours` is used to return a new list of all the neighbouring boids that the `boidToTest` object has (excluding itself). A boid is a neighbour if its x and y value is within the DETECT RADIUS of the `boidToTest` parameter. Make sure this class uses generics where appropriate and that it is thread safe.

   **(15 marks)**

| BoidFlock |
|---|
| +DETECTRADIUS : int<br>-boidList : List<Boid> |
| +BoidFlock()<br>+BoidFlock(numBoidsToStart:int)<br>+addBoidToFlock() : void<br>+removeBoidFromFlock() : void<br>+drawAllBoids(g:Graphics): void<br>+getNumberOfBoids() : int<br>+destroyAllBoids() : void<br>+getNeighbours(boidToTest:Boid) : List<Boid> |

A boid with co-ordinates (x0, y0) is a neighbour to another boid (x1, y1) if:

$$(x0 - x1)^2 + (y0 - y1)^2 < DETECTRADIUS^2$$

3. Prepare a simple GUI called `BoidGui` to test the `BoidFlock` and `Boid` class which should display an entire flock of boids that move around within a panel bouncing off the walls. The GUI should repeatedly update its display by drawing the boids at fixed time intervals.. Add buttons for adding and removing boids or clearing the screen. Add sliders for manipulating radius detection and separation, cohesion and alignment weights of the boids.

**(15 marks)**

4. Suppose an interface called `RandomObtainable` has been written for collections from which a random element should be obtainable. It has a method `getRandom` which should return an element randomly selected from the collection, and a method `removeRandom` that should try to remove a random element from the collection and `insertRandom` which inserts the element into the collection at a random location between zero and the size of the collection (inclusive). The `getRandom` should throw a `NoSuchElementException` if the collection is empty.

| RandomObtainable |
|---|
| |
| +getRandom() : E<br>+removeRandom() : boolean<br>+insertRandom(E element) : boolean |

Extend the `ArrayList` class so that it implements this interface, and test it with a driver main method. At the top of your class in the header comments state what the *asymptotic complexity* is for each of the methods and why? If instead your class extended `LinkedList`, what are the *asymptotic complexities*?

**(10 marks)**