Data Structures and Algorithms

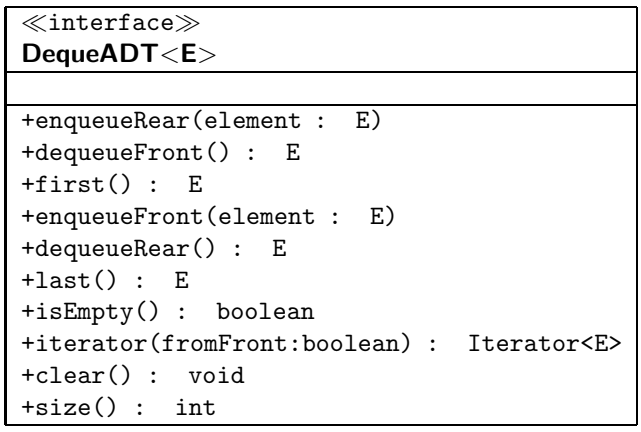
Assignment Two

Due Friday 22 April 2016

Ensure that it is submitted to AUTonline by 11:59pm on the due date.

Questions:

1. A *deque* (pronounced *deck*) or double-ended queue is a linear collection that allows insertion and deletion at both ends. Using the following interface called `DequeADT` available from AUTOnline, create an implementing class called `DoublyLinkedDeque<E extends Comparable>` which uses an underlying doubly linked list. You must create your own linked list and implement it using `Node<E>` objects. It has methods; `dequeueRear` and `dequeueFront` for removing an element from the front or rear of the deque, `enqueueRear` and `enqueueFront` for adding an element to the front or rear of the deque, `first` and `last` for getting the element at the front or rear of the deque, `isEmpty`, `first`, `clear` which empties the deque, and `iterator` which returns an object that implements the `Iterator` interface and should either navigate from the front or the back depending on the boolean parameter passed in. The interface also declares that a `NoSuchElementException` should be thrown if the deque is empty and a user tries to get or remove an element. Your class should also override a suitable `toString` for showing elements in deque from front to back. Include a suitable `main` which clearly demonstrates deque operations.



(15 marks)

2. Enhance `DoublyLinkedDeque` so that it contains a sort routine by adjusting node links. It should perform a recursive *quick sort* on the elements in the deque by using the `Comparable` elements `compareTo` method. Include a `main` which clearly demonstrates this algorithm. **(10 marks)**

3. Obtain the classes `Segment`, `Food` and the **unfinished** GUI called `SnakeGameGui` from AUTOnline which will be used to create a snake game which a snake can move around the world without crashing into the edge of the panel. The snake will eat food from a collection in order from smallest to biggest (determined by a food rating) and then grow by adding segments to its body. The snake will die if it doesn't eat the food in the right order or if a food that does not have the current lowest rating on screen crashes into the snake.

   - `Food` is used to describe an object which can run as a thread, the constructor takes in parameters for the width and height of the GUI as well as a *rating* which influences the size of the food. `Food` moves in random directions and stays within the bounds of the world. It also is assigned a random colour.
   - `Segment` is a class which describes a part of a `Snake` body. Its constructor takes in a starting x and y position for where it should be drawn, a size for how big the segment is, and a colour. The class also contains accessor and mutator methods for the x and y position of the segment as well as a `drawSegment` method which needs a `Graphics` object.
   - The unfinished `SnakeGameGui` class represents a canvas in which a `Snake` and a collection of `Food` can be drawn. It has a `Timer` for refreshing the innerclass `DrawPanel`, a simple button for restarting the game, and `KeyListener` which listens for the arrow keys.

   Alter the GUI so that it has a collection of `Food` objects each with an incremented rating held in a `DoublyLinkedDeque` (or use a `LinkedList` if you didn't get Question 1 going). The GUI should start the food up as threads and draw them to the display **(5 marks)**

4. Using the UML create a class called `Snake` which can move around a GUI and run in a thread. It takes in the width and height of the panel into its constructor. It holds a `DoublyLinkedDeque` (or use a `LinkedList` if you didn't get Question 1 going) of `Segment` objects which represent the snake body. It also has a boolean which sets the snake to be alive, an **enum** `Direction` which specifies which way the snake is moving, an integer *movement* value which determines how far a snake will move in that direction, an x and y position of the snake, and a size for the snake head (which is also used to set the `Segment` sizes). The `run` method should continuously loop while the snake is still alive and move the snake by calling `moveSnake`. While moving the snake needs to check to make sure it doesn't hit the edge of the panel otherwise it dies. It should update

its x or y coordinate depending on the direction that the head is facing. It also needs to iterate through its body segments updating each position with that of the previous segment (first segment goes to where the old head was). The method `eatFood` is used to check whether the food has come into range on the snakes head, if so then append new segments to the snake depending on rating and colour of the food (ie if snake eats blue food with rating 3, then add 3 blue segments), newer segments should appear near at the front of the snakes body. The `checkIfSnakeHit` method is used to pass in a food with a higher rating, it should return true if the food if it comes within range of its head or any of its segments allowing the GUI to then kill the snake. It also has methods to get the position of the snake, to check whether the snake is still alive, setting the direction of the snake (changed using keys from GUI), drawing at x, y position, and killing the snake (which terminates the thread). Care must be taken to ensure the program is thread safe. **(15 marks)**

Snake
-segments : DequeADT -alive : boolean -x,y : int -SIZE : int -movement : int + enum Direction$\{U, D, L, R\}$ - direction : Direction
+Snake(panelWidth : int, panelHeight :int) +run() : () -moveSnake() : void +isAlive() : boolean +killSnake() : void +getXPosition() : int +getYPosition() : int +setDirection(direciton : Direction) : E Void +checkIfSnakeHit(food : Food) : boolean +eatFoodIfInRange(food : Food) : Iterator<E> Boolean +drawSnake(g : Graphics) : void

$(snakeX - foodX)^2 + (snakeY - foodY)^2 < (FoodSize + SnakeSize)/2$

5. Alter the GUI further so that a `Snake` can now move around the screen and eat food in order, you will need to check conditions to determine if the snake is alive, that it eats the correct food and check that each higher rated food doesn't eat the snake (by crashing into its head or segments). Further enhance the game for up to 5 bonus marks, you could add more GUI components for controlling the game speed, or adding more food, or pictures, or alter snake so its head can't crash into its body, or add any other cool design or features. **(5 + 5 bonus marks)**